



Guía para desarrolladores

# AWS Lambda



# AWS Lambda: Guía para desarrolladores

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

---

# Table of Contents

¿Qué es AWS Lambda? .....	1
Cuándo debe utilizar Lambda .....	1
Características principales .....	2
Creación de su primera función .....	4
Requisitos previos .....	4
Cree una función de Lambda con la consola. ....	6
Invocar la función de Lambda mediante la consola .....	12
Limpieza .....	15
Recursos adicionales y próximos pasos .....	16
Creación de su primera aplicación .....	18
Requisitos previos .....	20
Descarga de los archivos de la aplicación de ejemplo .....	20
Implementación de la aplicación .....	28
Implementación de los recursos de forma manual .....	28
Implementación de los recursos mediante AWS SAM .....	37
Pruebas de la aplicación .....	38
Prueba de la aplicación de forma manual .....	39
Prueba de la aplicación con el script automatizado .....	41
Sigüientes pasos .....	42
Conceptos clave de Lambda .....	44
Función .....	44
Desencadenador .....	44
Evento .....	45
Entorno de ejecución .....	45
Paquete de implementación .....	46
Tiempo de ejecución .....	46
Capa .....	46
Simultaneidad .....	47
Qualifier .....	47
Destino .....	48
Infraestructura como código (IaC) .....	49
Herramientas de IaC para Lambda .....	49
Introducción a IaC para Lambda .....	51
Requisitos previos .....	52

Creación de una función de Lambda .....	52
Visualización de la plantilla de AWS SAM para su función .....	52
Uso de AWS Application Composer para diseñar una aplicación sin servidor .....	55
Implementación de su aplicación sin servidor mediante AWS SAM (opcional) .....	60
Puesta a prueba de la aplicación implementada (opcional) .....	63
Siguientes pasos .....	64
Regiones en las que se admite la integración de Lambda a Application Composer .....	64
Modelo de programación .....	66
Tiempos de ejecución de Lambda .....	68
Tiempos de ejecución admitidos .....	68
Nuevas versiones de tiempo de ejecución .....	71
Política de obsolescencia del tiempo de ejecución .....	71
Modelo de responsabilidad compartida .....	72
Uso del tiempo de ejecución después de la obsolescencia .....	74
Recepción de notificaciones de caducidad de un tiempo de ejecución .....	76
Tiempos de ejecución obsoletos .....	77
Actualizaciones de las versiones del tiempo de ejecución .....	81
Modos de actualización del tiempo de ejecución .....	82
Lanzamiento de la versión del tiempo de ejecución bifásico .....	83
Configuración de administración del tiempo de ejecución .....	84
Restauración a una versión de tiempo de ejecución .....	86
Identificación de los cambios de versión del tiempo de ejecución de Lambda .....	87
Modelo de responsabilidad compartida .....	89
Permisos .....	91
Obtener datos sobre las funciones por tiempo de ejecución .....	93
Lista de las versiones de funciones que utilizan un tiempo de ejecución determinado .....	93
Identificación de las funciones invocadas más recientemente y con mayor frecuencia .....	96
Modificaciones de tiempo de ejecución .....	100
Variables de entorno específicas del lenguaje .....	100
Scripts de encapsulador .....	100
API de tiempo de ejecución .....	104
Siguiente invocación .....	105
Respuesta de la invocación .....	106
Error de inicialización .....	106
Error de invocación .....	108
Tiempos de ejecución exclusivos del sistema operativo .....	110



Creación de un tiempo de ejecución personalizado .....	111
Tutorial de tiempo de ejecución personalizado .....	115
Entorno de ejecución .....	125
Ciclo de vida del entorno en tiempo de ejecución .....	126
Fase "init" .....	126
Errores durante la fase Init .....	127
Fase Restore (solo Lambda SnapStart) .....	128
Fase "invoke" .....	128
Errores durante la fase Invoke .....	129
Fase "shutdown" .....	132
Implementación de la ausencia de estado .....	133
Configuración de funciones de .....	134
archivos de archivo .zip .....	136
Creación de la función de Lambda .....	136
Uso del editor de código de la consola .....	138
Actualización del código de la función .....	138
Cambio del tiempo de ejecución .....	139
Modificación de la arquitectura .....	140
Uso de la API de Lambda .....	140
AWS CloudFormation .....	141
Imágenes de contenedor .....	142
Requisitos .....	143
Uso de una imagen base de AWS .....	144
Uso de una imagen base exclusiva del sistema operativo de AWS .....	145
Uso de una imagen base que no es de AWS .....	146
Clientes de interfaz de tiempo de ejecución .....	146
Permisos de Amazon ECR .....	147
Ciclo de vida de la función .....	150
Memoria .....	151
Cuándo aumentar la memoria .....	151
Uso de la consola .....	152
Uso de la AWS CLI .....	152
Uso de AWS SAM .....	153
Aceptación de recomendaciones de memoria de función (consola) .....	153
Almacenamiento efímero .....	154
Casos de uso .....	154

Uso de la consola .....	155
Uso de la AWS CLI .....	155
Uso de AWS SAM .....	156
Conjuntos de instrucciones (ARM/x86) .....	157
Ventajas del uso de la arquitectura arm64 .....	157
Requisitos para migrar a la arquitectura arm64 .....	158
Compatibilidad de código de función con la arquitectura arm64 .....	158
Cómo migrar a la arquitectura arm64 .....	159
Configuración de la arquitectura del conjunto de instrucciones .....	160
Tiempo de espera .....	161
Cuándo aumentar el tiempo de espera .....	161
Mediante la consola .....	162
Uso de la AWS CLI .....	162
Uso de AWS SAM .....	162
Configuración de variables de entorno .....	164
Variables definidas de entorno de tiempo de ejecución .....	168
Escenario de ejemplo para variables de entorno .....	170
Proteger variables de entorno .....	170
Recuperar variables de entorno .....	174
Vinculación de funciones a una VPC .....	176
Permisos de IAM necesarios .....	177
Conexión de las funciones de Lambda a una Amazon VPC en su Cuenta de AWS .....	178
Acceso a Internet cuando está conectado a una VPC .....	182
Compatibilidad con IPv6 .....	182
Prácticas recomendadas para utilizar Lambda con Amazon VPC .....	183
Introducción a las interfaces de red elástica (ENI) de hiperplano .....	185
Uso de claves de condición de IAM para la configuración de la VPC .....	186
Tutoriales de VPC .....	190
Acceso a Internet para funciones de VPC .....	191
Redes entrantes .....	217
Consideraciones para los puntos de enlace de la interfaz Lambda .....	217
Creación de un punto de enlace de interfaz para Lambda .....	218
Creación de una política de punto de enlace de interfaz para Lambda .....	220
Sistema de archivos .....	222
Permisos de usuario y rol de ejecución .....	222
Configuración de un sistema de archivos y un punto de acceso .....	223

Conexión a un sistema de archivos (consola) .....	224
Alias .....	226
Uso de alias .....	228
Configuraciones de direccionamiento .....	229
Versiones .....	232
Creación de versiones de funciones .....	233
Uso de versiones .....	234
Concesión de permisos .....	235
Etiquetas .....	236
Permisos .....	236
Uso de etiquetas con la consola .....	236
Uso de etiquetas con la AWS CLI .....	239
Requisitos para etiquetas .....	240
Transmisión de respuestas .....	242
Límites de ancho de banda para la transmisión de respuestas .....	243
Escritura de las funciones .....	243
Invocación de funciones de .....	245
Tutorial: Creación de una función de transmisión de respuesta con una URL de función .....	247
Invocación de funciones de .....	251
Invocación de una función de forma sincrónica .....	253
Invocación asincrónica .....	257
Invocación de una función de Lambda de forma asíncrona .....	257
Administración de errores de invocación asíncrona .....	258
Configuración de la invocación asíncrona .....	259
Captura de registros de invocaciones asíncronas .....	261
Mapeos de origen de eventos .....	270
Asignaciones de orígenes de eventos y desencadenadores .....	270
Comportamiento de procesamiento por lotes .....	271
API de asignación de orígenes de eventos .....	274
Filtrado de eventos .....	275
Conceptos básicos del filtrado de eventos .....	276
Gestión de registros que no cumplen con los criterios de filtro .....	278
Sintaxis de la regla de filtro .....	279
Adjuntar criterios de filtro a una asignación de origen de eventos (consola) .....	281
Adjuntar criterios de filtro a una asignación de origen de eventos (AWS CLI) .....	282
Adjuntar criterios de filtro a una asignación de origen de eventos (AWS SAM) .....	283

Uso de filtros con diferentes Servicios de AWS .....	284
Pruebas en la consola .....	286
Invocación de funciones con eventos de prueba .....	286
Creación de eventos de prueba privados .....	287
Creación de eventos de prueba compartibles .....	287
Eliminación de esquemas de eventos de prueba compartibles .....	289
Estados de la función .....	290
Estados de función durante la actualización .....	291
Reintentos .....	293
Detección de bucles recursivos .....	295
Descripción de la detección de bucles recursivos .....	296
Servicios de AWS y SDK admitidos .....	297
Notificaciones de bucle recursivo .....	300
Cómo responder a las notificaciones de detección de bucles recursivos .....	301
URL de funciones .....	303
Creación de una URL de función (consola) .....	304
Creación de una URL de función (AWS CLI) .....	306
Adición de una URL de función a una plantilla de CloudFormation .....	307
Uso compartido de recursos entre orígenes (CORS) .....	308
URL de funciones de limitación .....	310
Desactivación de URL de funciones .....	310
Eliminación de URL de función .....	310
Control de acceso .....	311
Invocación de URL de funciones .....	320
Supervisión de las URL de funciones .....	332
Tutorial: Creación de una función con una URL de función .....	334
Escalado de funciones .....	340
Cómo comprender y visualizar la simultaneidad .....	340
Calcular la simultaneidad de una función .....	345
Comprender la simultaneidad reservada y simultaneidad aprovisionada .....	347
Simultaneidad reservada .....	347
Simultaneidad aprovisionada .....	350
Cómo asigna Lambda la simultaneidad aprovisionada .....	355
Comparación de la simultaneidad reservada con la aprovisionada .....	355
Descripción de la simultaneidad y las solicitudes por segundo .....	356
Cuotas de simultaneidad .....	358

Configuración de la simultaneidad reservada .....	361
Configuración de la simultaneidad reservada .....	362
Estimar con precisión la simultaneidad reservada requerida para una función .....	363
Configuración de simultaneidad aprovisionada .....	365
Configuración de simultaneidad aprovisionada .....	366
Estimar con precisión la simultaneidad aprovisionada requerida para una función .....	368
Optimizar el código de la función cuando se utiliza la simultaneidad aprovisionada .....	369
Usar variables de entorno para ver y controlar el comportamiento de simultaneidad aprovisionado .....	370
Comprenda el comportamiento de registro y facturación con la simultaneidad aprovisionada .....	370
Uso de Application Auto Scaling para automatizar la administración de simultaneidad aprovisionada .....	371
Comportamiento del escalado .....	375
Tasa de escalado de simultaneidad .....	375
Monitoreo de la simultaneidad .....	377
Métricas generales de simultaneidad .....	377
Métricas de simultaneidad aprovisionada .....	377
Cómo trabajar con la métrica ClaimedAccountConcurrency .....	380
Compilación con Node.js .....	383
Inicialización de Node.js .....	386
Designación de un controlador de funciones como módulo de ES .....	386
Versiones del SDK incluidas en el tiempo de ejecución .....	387
Uso de keep-alive .....	387
Carga del certificado de la CA .....	388
Controlador .....	389
Conceptos básicos del controlador de Node.js .....	389
Denominación .....	390
Uso de async/await .....	391
Uso de devolución de llamadas .....	393
Prácticas recomendadas de codificación para las funciones de Lambda en Node.js .....	396
Implementar archivos de archivos .zip .....	399
Dependencias de tiempo de ejecución en Node.js .....	399
Creación de un paquete de despliegue .zip sin dependencias .....	400
Creación de un paquete de despliegue .zip con dependencias .....	400
Creación de una capa de Node.js para las dependencias .....	402

Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución .....	403
Creación y actualización de funciones de Lambda Node.js mediante archivos .zip .....	404
Implementación de imágenes de contenedor .....	411
AWS imágenes base para Node.js .....	412
Uso de una imagen base de AWS .....	413
Uso de una imagen base que no sea de AWS .....	419
Context .....	429
Registro .....	431
Crear una función que devuelve registros .....	431
Uso de controles de registro avanzados de Lambda con Node.js .....	433
Visualización de los registros en la consola de Lambda .....	440
Visualización de los registros de en la consola de CloudWatch .....	440
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	440
Eliminación de registros .....	444
Rastreo .....	445
Uso de ADOT para instrumentar las funciones de Node.js .....	446
Uso del SDK de X-Ray para instrumentar las funciones de Node.js .....	446
Activación del seguimiento con la consola de Lambda .....	447
Activación del seguimiento con la API de Lambda .....	448
Activación del seguimiento con AWS CloudFormation .....	448
Interpretación de un seguimiento de X-Ray .....	449
Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK) .....	452
Compilación con TypeScript .....	454
Entorno de desarrollo .....	455
Controlador .....	457
Conceptos básicos del controlador de Typescript .....	457
Uso de async/await .....	458
Uso de devolución de llamadas .....	459
Uso de tipos para el objeto event .....	461
Prácticas recomendadas de codificación para las funciones de Lambda en Typescript .....	462
Implementar archivos de archivos .zip .....	465
Uso de AWS SAM .....	465
Utilización del AWS CDK .....	467
Uso de AWS CLI y esbuild .....	470
Implementación de imágenes de contenedor .....	473

Uso de una imagen base de Node.js para compilar y empaquetar código de función de TypeScript .....	473
Context .....	481
Registro .....	483
Herramientas y bibliotecas .....	483
Uso de Powertools para AWS Lambda (TypeScript) y AWS SAM para el registro estructurado .....	484
Uso de Powertools para AWS Lambda (TypeScript) y el AWS CDK para el registro estructurado .....	487
Visualización de los registros en la consola de Lambda .....	490
Visualización de los registros de en la consola de CloudWatch .....	491
Rastreo .....	492
Uso de Powertools para AWS Lambda (TypeScript) y AWS SAM para el seguimiento .....	493
Uso de Powertools para AWS Lambda (TypeScript) y el AWS CDK para el seguimiento .....	495
Interpretación de un seguimiento de X-Ray .....	499
Compilación con Python .....	500
Versiones del SDK incluidas en el tiempo de ejecución .....	502
Formato de respuesta .....	502
Cierre correcto de las extensiones .....	503
Controlador .....	504
Denominación .....	504
Funcionamiento .....	505
Devolver un valor .....	505
Ejemplos .....	506
Prácticas recomendadas de codificación para las funciones de Lambda en Python .....	508
Implementar archivos de archivos .zip .....	511
Dependencias de tiempo de ejecución en Python .....	511
Creación de un paquete de despliegue .zip sin dependencias .....	512
Creación de un paquete de despliegue .zip con dependencias .....	513
Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución .....	516
Uso de carpetas <code>__pycache__</code> .....	517
Creación de paquetes de despliegue .zip con bibliotecas nativas .....	517
Creación y actualización de funciones de Lambda en Python mediante archivos .zip .....	519
Implementación de imágenes de contenedor .....	526
AWS imágenes base para Python .....	527
Uso de una imagen base de AWS .....	529

Uso de una imagen base que no sea de AWS .....	535
Capas .....	545
Requisitos previos .....	545
Compatibilidad de capas de Python con Amazon Linux .....	546
Rutas de capa para tiempos de ejecución de Python .....	547
Empaquetado del contenido de la capa .....	548
Creación de la capa .....	549
Agregue la capa a la función .....	550
Trabajo con ruedas de distribuciones <code>manylinux</code> .....	553
Context .....	558
Registro y supervisión de las funciones de Lambda de Python .....	560
Impresión en el registro .....	560
Uso de una biblioteca de registro .....	561
Uso de los controles de registro avanzados de Lambda con Python .....	563
Visualización de los registros en la consola de Lambda .....	568
Visualización de los registros en la consola de CloudWatch .....	568
Visualización de los registros con AWS CLI .....	568
Eliminación de registros .....	572
Herramientas y bibliotecas .....	572
Uso de Powertools para AWS Lambda (Python) y AWS SAM para el registro estructurado .	572
Uso de Powertools para AWS Lambda (Python) y AWS CDK para el registro estructurado .	576
Pruebas .....	583
Pruebas de aplicaciones sin servidor .....	584
Rastreo .....	586
Uso de Powertools para AWS Lambda (Python) y AWS SAM para el seguimiento .....	587
Uso de Powertools para AWS Lambda (Python) y el AWS CDK para el seguimiento .....	590
Uso de ADOT para instrumentar las funciones de Python .....	595
Uso del X-Ray SDK para instrumentar las funciones de Python .....	595
Activación del seguimiento con la consola de Lambda .....	596
Activación del seguimiento con la API de Lambda .....	596
Activación del seguimiento con AWS CloudFormation .....	597
Interpretación de un seguimiento de X-Ray .....	597
Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK) .....	600
Compilación con Ruby .....	602
Versiones del SDK incluidas en el tiempo de ejecución .....	604
Habilitación de otro JIT de Ruby (YJIT) .....	604



Controlador .....	606
Concepts básicos del controlador de Ruby .....	606
Prácticas recomendadas de codificación para las funciones de Lambda en Ruby .....	607
Implementar archivos de archivos .zip .....	610
Dependencias en Ruby .....	611
Creación de un paquete de despliegue .zip sin dependencias .....	611
Creación de un paquete de despliegue .zip con dependencias .....	611
Creación de una capa de Ruby para las dependencias .....	613
Creación de paquetes de despliegue .zip con bibliotecas nativas .....	614
Creación y actualización de funciones de Lambda Ruby mediante archivos .zip .....	616
Implementación de imágenes de contenedor .....	623
Imágenes base de AWS para Ruby .....	624
Uso de una imagen base de AWS .....	624
Uso de una imagen base que no sea de AWS .....	631
Context .....	640
Registro .....	641
Crear una función que devuelve registros .....	641
Visualización de los registros en la consola de Lambda .....	643
Visualización de los registros de en la consola de CloudWatch .....	643
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	643
Eliminación de registros .....	647
Trabajo con la biblioteca de registradores en Ruby .....	647
Rastreo .....	649
Habilitación del seguimiento activo con la API de Lambda .....	655
Habilitación del seguimiento activo con AWS CloudFormation .....	655
Almacenamiento de dependencias en tiempo de ejecución en una capa .....	656
Compilación con Java .....	657
Controlador .....	660
Ejemplo de controlador: tiempos de ejecución de Java 17 .....	660
Ejemplo de controlador: tiempos de ejecución de Java 11 e inferiores .....	662
Código de inicialización .....	663
Elección de los tipos de entrada y salida .....	664
Interfaces de controlador .....	665
Prácticas recomendadas de codificación para las funciones de Lambda .....	667
Ejemplo de código del controlador .....	670
Implementar archivos de archivos .zip .....	672

Requisitos previos .....	672
Herramientas y bibliotecas .....	672
Compilación de un paquete de implementación con Gradle .....	674
Creación de una capa de Java para las dependencias .....	675
Compilación de un paquete de implementación con Maven .....	676
Carga de un paquete de despliegue con la consola de Lambda .....	679
Carga de un paquete de despliegue con la AWS CLI .....	680
Carga de un paquete de implementación con AWS SAM .....	682
Implementación de imágenes de contenedor .....	685
Imágenes base de AWS para Java .....	686
Uso de una imagen base de AWS .....	687
Uso de una imagen base que no sea de AWS .....	696
Capas .....	707
Requisitos previos .....	707
Compatibilidad de capas de Java con Amazon Linux .....	708
Rutas de capa para tiempos de ejecución de Java .....	708
Empaquetado del contenido de la capa .....	709
Creación de la capa .....	711
Agregue la capa a la función .....	711
Lambda SnapStart .....	716
Características y limitaciones compatibles .....	717
Regiones admitidas .....	717
Consideraciones sobre compatibilidad .....	718
Precios .....	719
SnapStart y simultaneidad aprovisionada .....	719
Recursos adicionales de .....	720
Activando SnapStart .....	721
Control de la exclusividad .....	727
Enlaces de tiempo de ejecución .....	730
Supervisión .....	734
Modelo seguridad .....	737
Prácticas recomendadas .....	738
Personalización de Java .....	742
Cómo entender la variable de entorno JAVA_TOOL_OPTIONS .....	742
Context .....	745
Contexto en aplicaciones de ejemplo .....	747

Registro .....	749
Crear una función que devuelve registros .....	749
Uso de los controles de registro avanzados de Lambda con Java .....	751
Implementación del registro avanzado con Log4j2 y SLF4J .....	754
Herramientas y bibliotecas .....	758
Uso de Powertools para AWS Lambda (Java) y AWS SAM para el registro estructurado .....	758
Visualización de los registros en la consola de Lambda .....	763
Visualización de los registros de en la consola de CloudWatch .....	763
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	764
Eliminación de registros .....	767
Código de registro de ejemplo .....	767
Rastreo .....	769
Uso de Powertools para AWS Lambda (Java) y AWS SAM para el seguimiento .....	770
Uso de Powertools para AWS Lambda (Java) y el AWS CDK para el seguimiento .....	772
Uso de ADOT para instrumentar las funciones de Java .....	784
Uso del X-Ray SDK para instrumentar las funciones de Java .....	784
Activación del seguimiento con la consola de Lambda .....	785
Activación del seguimiento con la API de Lambda .....	785
Activación del seguimiento con AWS CloudFormation .....	786
Interpretación de un seguimiento de X-Ray .....	786
Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK) .....	789
Seguimiento de X-Ray en aplicaciones de ejemplo (X-Ray SDK) .....	790
Aplicaciones de ejemplo .....	792
Compilación con Go .....	794
Compatibilidad del tiempo de ejecución de Go .....	794
Herramientas y bibliotecas .....	795
Controlador .....	797
Conceptos básicos del controlador de Go .....	797
Denominación .....	799
Controlador de funciones de Lambda con tipos estructurados .....	799
Uso del estado global .....	801
Prácticas recomendadas de codificación para las funciones de Lambda en Go .....	803
Context .....	805
Acceso a la información del contexto de invocación .....	805
Implementar archivos de archivos .zip .....	808
Creación de un archivo.zip en macOS y Linux .....	808

Crear un archivo.zip en Windows .....	810
Creación y actualización de funciones de Lambda en Go mediante archivos .zip .....	813
Creación de una capa de Go para las dependencias .....	819
Implementación de imágenes de contenedor .....	821
Imágenes base de AWS para implementar funciones de Go .....	821
Cliente de interfaz de tiempo de ejecución de Go .....	822
Uso de una imagen base exclusiva del sistema operativo de AWS .....	822
Uso de una imagen base que no sea de AWS .....	829
Registro .....	838
Crear una función que devuelve registros .....	838
Visualización de los registros en la consola de Lambda .....	840
Visualización de los registros de en la consola de CloudWatch .....	840
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	841
Eliminación de registros .....	844
Rastreo .....	845
Uso de ADOT para instrumentar las funciones de Go .....	846
Uso del SDK de X-Ray para instrumentar las funciones de Go .....	846
Activación del seguimiento con la consola de Lambda .....	846
Activación del seguimiento con la API de Lambda .....	847
Activación del seguimiento con AWS CloudFormation .....	847
Interpretación de un seguimiento de X-Ray .....	848
Compilación con C# .....	852
Entorno de desarrollo .....	852
Instalación de las plantillas del proyecto .NET .....	852
Instalación y actualización de las herramientas de la CLI .....	853
Controlador .....	854
Modelos de ejecución de .NET para Lambda .....	854
Controladores de bibliotecas de clases .....	855
Controladores de conjuntos ejecutables .....	856
Serialización de las funciones de Lambda .....	857
Simplifique el código de funciones con el marco de anotaciones Lambda .....	859
Restricciones del controlador de funciones de Lambda .....	862
Prácticas recomendadas de codificación para las funciones de Lambda en C# .....	862
Paquete de implementación .....	865
CLI de NET Lambda Global .....	866
AWS SAM .....	872

AWS CDK .....	876
ASP.NET .....	879
Implementación de imágenes de contenedor .....	885
AWS imágenes base para .NET .....	886
Uso de una imagen base de AWS .....	886
Uso de una imagen base que no sea de AWS .....	889
Compilación nativa anticipada .....	893
Tiempo de ejecución de Lambda .....	893
Requisitos previos .....	894
Introducción .....	895
Serialización .....	898
Recorte .....	898
Resolución de problemas .....	899
Context .....	900
Registro .....	902
Crear una función que devuelve registros .....	902
Herramientas y bibliotecas .....	903
Uso de Powertools para AWS Lambda (.NET) y AWS SAM para el registro estructurado ....	903
Visualización de los registros en la consola de Lambda .....	906
Visualización de los registros de en la consola de CloudWatch .....	907
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	907
Eliminación de registros .....	910
Rastreo .....	911
Uso de Powertools para AWS Lambda (.NET) y AWS SAM para el seguimiento .....	912
Uso del SDK de X-Ray para instrumentar las funciones de .NET .....	915
Activación del seguimiento con la consola de Lambda .....	916
Activación del seguimiento con la API de Lambda .....	917
Activación del seguimiento con AWS CloudFormation .....	917
Interpretación de un seguimiento de X-Ray .....	918
Pruebas .....	921
Pruebas de aplicaciones sin servidor .....	922
Compilación con PowerShell .....	926
Entorno de desarrollo .....	928
Paquete de implementación .....	929
Creación de una función de Lambda .....	929
Controlador .....	932

Devolución de datos .....	933
Context .....	934
Registro .....	935
Crear una función que devuelve registros .....	935
Visualización de los registros en la consola de Lambda .....	937
Visualización de los registros de en la consola de CloudWatch .....	937
Visualización de los registros mediante la AWS Command Line Interface (AWS CLI) .....	938
Eliminación de registros .....	941
Compilación con Rust .....	942
Controlador .....	944
Conceptos básicos del controlador de Rust .....	944
Uso del estado compartido .....	945
Prácticas recomendadas de codificación para las funciones de Lambda en Rust .....	946
Context .....	949
Acceso a la información del contexto de invocación .....	949
Eventos HTTP .....	951
Implementar archivos de archivos .zip .....	954
Requisitos previos .....	954
Creación de la función .....	954
Implementación de la función .....	955
Invocación de la función .....	957
Registro .....	959
Creación de una función que escribe registros .....	959
Implementación del registro avanzado con la caja Tracing .....	960
Prácticas recomendadas .....	962
Código de función .....	962
Función de configuración .....	964
Escalabilidad de las funciones .....	965
Métricas y alarmas .....	966
Uso de secuencias .....	966
Prácticas recomendadas de seguridad .....	967
Prueba de funciones sin servidor .....	969
Resultados empresariales específicos .....	970
Qué se debe probar .....	970
Cómo efectuar las pruebas sin servidor .....	971
Técnicas de pruebas .....	972

Pruebas en la nube .....	973
Pruebas con simulaciones .....	976
Pruebas con emulación .....	977
Prácticas recomendadas .....	978
Priorice las pruebas en la nube .....	978
Estructure su código para que sea fácil de probar .....	979
Acelere los ciclos de retroalimentación del desarrollo .....	979
Céntrese en las pruebas de integración .....	979
Cree entornos de pruebas aislados .....	980
Utilice simulaciones para una lógica empresarial aislada .....	981
Use emuladores con moderación .....	982
Desafíos al probar de forma local .....	982
Ejemplo: la función de Lambda crea un bucket de S3 .....	983
Ejemplo: la función de Lambda procesa mensajes de una cola de Amazon SQS .....	983
Preguntas frecuentes .....	984
Próximos pasos y recursos .....	985
Integración de otros servicios .....	987
Creación de un desencadenador .....	987
Lista de servicios .....	988
Apache Kafka .....	991
Evento de ejemplo .....	992
Configuración de orígenes de eventos .....	993
Procesamiento de mensajes .....	1002
Filtrado de eventos .....	1010
Destinos en caso de error .....	1016
Resolución de problemas .....	1020
API Gateway .....	1024
Elegir un tipo de API .....	1024
Adición de un punto de conexión a la función de Lambda .....	1027
Integración de proxy .....	1027
Formato de eventos .....	1028
Formato de respuesta .....	1029
Permisos .....	1030
Aplicación de muestra .....	1032
Tutorial .....	1032
Errores .....	1054

Application Composer .....	1055
Exportación de la función de Lambda a Application Composer .....	1056
Otros recursos .....	1058
CloudFormation .....	1059
CloudFront (Lambda @Edge) .....	1063
DocumentDB .....	1065
Ejemplo de evento de Amazon DocumentDB .....	1066
Requisitos previos y permisos .....	1067
Configuración de red .....	1069
Creación de una asignación de orígenes de eventos de Amazon DocumentDB (consola) .	1071
Creación de una asignación de orígenes de eventos de Amazon DocumentDB (SDK o	
CLI) .....	1073
Posiciones iniciales de flujos y sondeo .....	1076
Monitoreo del origen de eventos de Amazon DocumentDB .....	1076
Tutorial .....	1077
DynamoDB .....	1112
Sondeo y procesamiento por lotes de flujos .....	1112
Posiciones iniciales de flujos y sondeo .....	1114
Lectores simultáneos .....	1114
Evento de ejemplo .....	1114
Creación de asignaciones .....	1116
Fallos de elementos por lotes .....	1118
Control de errores .....	1132
Supervisión .....	1135
Procesamiento con estado .....	1135
Parámetros .....	1140
Filtrado de eventos .....	1143
Tutorial .....	1152
EC2 .....	1169
Cómo otorgar permisos a EventBridge (Eventos de CloudWatch) .....	1170
Elastic Load Balancing (Equilibrador de carga de aplicación) .....	1171
Invocar mediante un programador de EventBridge .....	1174
Configurar el rol de ejecución .....	1174
Crear una programación .....	1174
Recursos relacionados .....	1179
IoT .....	1180



Kinesis Data Streams .....	1182
Flujos de sondeo y procesamiento por lotes .....	1183
Evento de ejemplo .....	1184
Creación de asignaciones .....	1185
Fallos de elementos por lotes .....	1191
Control de errores .....	1206
Procesamiento con estado .....	1209
Parámetros .....	1213
Filtrado de eventos .....	1215
Tutorial .....	1220
MQ .....	1237
Comprender el grupo de consumidores de Lambda para Amazon MQ .....	1239
Creación de asignaciones .....	1243
Parámetros .....	1250
Filtrado de eventos .....	1251
Solución de problemas .....	1257
MSK .....	1259
Evento de ejemplo .....	1260
Configuración de orígenes de eventos .....	1261
Procesamiento de mensajes .....	1274
Filtrado de eventos .....	1283
Destinos en caso de error .....	1289
Tutorial .....	1293
RDS .....	1308
Configuración de la función para que funcione con los recursos de RDS .....	1308
Conexión a una base de datos de Amazon RDS en una función de Lambda .....	1311
Procesamiento de las notificaciones de eventos de Amazon RDS .....	1327
Tutorial completo de Lambda y Amazon RDS .....	1328
S3 .....	1329
Tutorial: Utilizar un trigger S3 .....	1331
Tutorial: Uso de un desencadenador de Amazon S3 para crear miniaturas .....	1358
SQS .....	1388
Descripción del comportamiento de sondeo y procesamiento por lotes para las asignaciones de orígenes de eventos de Amazon SQS .....	1388
Ejemplo de evento de mensaje en cola estándar .....	1389
Ejemplo de evento de mensajes de cola FIFO .....	1391

Creación de asignaciones .....	1392
Comportamiento del escalado .....	1396
Control de errores .....	1398
Parámetros .....	1411
Filtrado de eventos .....	1412
Tutorial .....	1417
Tutorial entre cuentas de SQS .....	1437
Lote S3 .....	1444
Invocación de una función de Lambda desde operaciones por lotes de Amazon S3 .....	1445
SNS .....	1447
Adición de un desencadenador de temas de Amazon SNS para una función de Lambda mediante la consola .....	1448
Adición manual de un desencadenador de temas de Amazon SNS para una función de Lambda .....	1448
Ejemplo de forma de evento SNS .....	1449
Tutorial .....	1450
Permisos de Lambda .....	1472
Rol de ejecución (permisos para que las funciones accedan a otros recursos) .....	1474
Creación de un rol de ejecución en la consola de IAM .....	1474
Creación y administración de roles con la AWS CLI .....	1475
Otorgue privilegios de acceso mínimos a su rol de ejecución de Lambda .....	1477
Actualización de un rol de ejecución .....	1477
Políticas administradas de AWS .....	1479
ARN de la función de origen .....	1482
Permisos de acceso (permisos para que otras entidades accedan a sus funciones) .....	1487
Políticas basadas en identidad .....	1487
Políticas basadas en recursos .....	1494
Control de acceso basado en atributos .....	1502
Recursos y condiciones .....	1509
Seguridad, gobernanza y conformidad .....	1520
Protección de los datos .....	1521
Cifrado en tránsito .....	1522
Cifrado en reposo .....	1522
Identity and Access Management .....	1523
Público .....	1523
Autenticación con identidades .....	1524

Administración de acceso mediante políticas .....	1528
Cómo AWS Lambda funciona con IAM .....	1530
Ejemplos de políticas basadas en identidades .....	1537
Políticas administradas de AWS .....	1541
Resolución de problemas .....	1547
Gobernanza .....	1549
Controles proactivos con Guard .....	1551
Controles proactivos con AWS Config .....	1555
Controles de detección con AWS Config .....	1563
Firma de código .....	1568
Análisis de código .....	1571
Observabilidad .....	1576
Validación de conformidad .....	1585
Resiliencia .....	1585
Seguridad de la infraestructura .....	1586
Firma de código .....	1587
Validación de firmas .....	1588
Configuración de la firma de código con la API de Lambda .....	1589
Creación de una configuración .....	1589
Actualización de la configuración .....	1591
Permisos .....	1592
Supervisión de funciones .....	1594
Precios .....	1594
Consola de administración .....	1595
Uso de la consola de Lambda .....	1595
Tipos de gráficos de monitorización .....	1595
Visualización de gráficos en la consola de Lambda .....	1596
Visualización de consultas en la consola de CloudWatch Logs .....	1597
Sigüientes pasos .....	1598
Ver métricas de funciones .....	1599
Visualización de métricas en la consola de CloudWatch .....	1599
Tipos de métricas .....	1600
Registros de funciones .....	1605
Permisos de IAM necesarios .....	1605
Precios .....	1606
Configuración de registros de funciones .....	1606

Visualización de los registros de función .....	1621
Registros de CloudTrail .....	1626
Eventos de datos de Lambda en CloudTrail .....	1627
Eventos de administración de Lambda en CloudTrail .....	1629
Uso de CloudTrail para solucionar problemas de orígenes de eventos de Lambda deshabilitados .....	1631
Ejemplos de eventos de Lambda .....	1632
AWS X-Ray .....	1635
Comprensión de los rastros .....	1636
Permisos de rol de ejecución .....	1641
El daemon de AWS X-Ray .....	1641
Habilitación del seguimiento activo con la API de Lambda .....	1641
Habilitación del seguimiento activo con AWS CloudFormation .....	1642
Información de las funciones .....	1643
Funcionamiento .....	1643
Precios .....	1644
Tiempos de ejecución admitidos .....	1644
Activación de Lambda Insights en la consola .....	1644
Habilitación de Lambda Insights mediante programación .....	1645
Uso del panel de información de Lambda Insights .....	1645
Detección de anomalías de función .....	1647
Solución de problemas de una función .....	1649
Sigüientes pasos .....	1598
Capas de Lambda .....	1652
Cómo usar las capas .....	1654
Capas y versiones de capas .....	1654
Empaquetado de las capas .....	1655
Rutas de capa para cada tiempo de ejecución de Lambda .....	1655
Creación y eliminación de capas .....	1659
Creación de una capa .....	1659
Eliminación de una versión de capa .....	1661
Adición de capas .....	1662
Acceso al contenido de la capa desde la función .....	1664
Búsqueda de información de capa .....	1664
Capas con AWS CloudFormation .....	1667
Capas con AWS SAM .....	1668

Extensiones de Lambda .....	1669
Entorno de ejecución .....	1670
Impacto de desempeño y recursos .....	1671
Permisos .....	1672
Configuración de extensiones .....	1673
Configuración de extensiones (archivo de archivo .zip) .....	1673
Uso de extensiones en imágenes de contenedor .....	1673
Sigüientes pasos .....	1674
Socios de extensiones .....	1675
Extensiones administradas de AWS .....	1676
API de extensiones .....	1677
Ciclo de vida del entorno de ejecución de Lambda .....	1678
Referencia de la API de extensiones .....	1688
API de telemetría .....	1694
Creación de extensiones mediante la API de telemetría .....	1695
Registro de sus extensiones .....	1697
Creación de un oyente de telemetría .....	1698
Especificación de un protocolo de destino .....	1699
Configuración del uso de memoria y el almacenamiento en búfer .....	1700
Envío de una solicitud de suscripción a la API de telemetría .....	1702
Mensajes entrantes de la API de telemetría .....	1703
Referencia de la API .....	1706
Referencia del esquema Event .....	1710
Conversión de eventos a OTel Spans .....	1731
API de registros .....	1738
Resolución de problemas .....	1751
Implementación .....	1751
General: Se deniega el permiso/No se puede cargar dicho archivo .....	1752
General: se produce un error al llamar al UpdateFunctionCode .....	1753
Amazon S3: Código de error PermanentRedirect. ....	1753
General: No se puede encontrar, no se puede cargar, no se puede importar, clase no encontrada, ningún archivo o directorio .....	1753
General: controlador de método no definido .....	1754
Lambda: error de conversión de la capa .....	1755
Lambda: InvalidParameterValueException o RequestEntityTooLargeException .....	1755
Lambda: InvalidParameterValueException .....	1756

Lambda: cuotas de simultaneidad y memoria .....	1756
Invocación .....	1757
IAM: lambda:InvokeFunction no autorizado .....	1757
Lambda: no se encontró un arranque válido (Runtime.InvalidEntrypoint) .....	1757
Lambda: no se puede realizar la operación ResourceConflictException .....	1758
Lambda: la función está atascada en Pendiente .....	1758
Lambda: una función utiliza toda la concurrencia .....	1758
General: no se puede invocar la función con otras cuentas o servicios .....	1759
General: la invocación de funciones está en bucle .....	1759
Lambda: enrutamiento de alias con concurrencia aprovisionada .....	1759
Lambda: comienza en frío con concurrencia aprovisionada .....	1759
Lambda: el frío comienza con nuevas versiones .....	1760
EFS: la función no pudo montar el sistema de archivos EFS .....	1761
EFS: la función no se pudo conectar al sistema de archivos EFS .....	1761
EFS: La función no pudo montar el sistema de archivos EFS debido al tiempo de espera .	1761
Lambda: Lambda detectó un proceso de E/S que estaba tomando demasiado tiempo .....	1762
Ejecución .....	1762
Lambda: la ejecución lleva demasiado tiempo .....	1762
Lambda: los registros o rastros no aparecen .....	1763
Lambda: no aparecen todos los registros de mi función .....	1763
Lambda: la función regresa antes de que finalice la ejecución .....	1764
AWS SDK: versiones y actualizaciones .....	1765
Python: las bibliotecas se cargan de manera incorrecta .....	1765
Red .....	1766
VPC: La función pierde acceso a Internet o agota el tiempo de espera .....	1766
VPC: la función necesita acceso a los servicios de AWS sin utilizar internet .....	1766
VPC: se ha alcanzado el límite de la interfaz de red elástica .....	1767
EC2: interfaz de red elástica con tipo de "lambda" .....	1767
DNS: no se puede conectar a los hosts con UNKNOWNHOSTEXCEPTION .....	1767
Aplicaciones de Lambda .....	1769
Monitorización de aplicaciones .....	1770
Implementaciones continuas .....	1772
Ejemplo AWS SAM de la plantilla Lambda .....	1773
Kubernetes .....	1775
Controladores para Kubernetes (ACK) de AWS .....	1775
Crossplane .....	1776

Aplicaciones de muestra .....	1777
Uso de los AWS SDK .....	1780
Ejemplos de código .....	1782
Acciones .....	1794
CreateAlias .....	1795
CreateFunction .....	1797
DeleteAlias .....	1817
DeleteFunction .....	1818
DeleteFunctionConcurrency .....	1830
DeleteProvisionedConcurrencyConfig .....	1831
GetAccountSettings .....	1832
GetAlias .....	1833
GetFunction .....	1835
GetFunctionConcurrency .....	1844
GetFunctionConfiguration .....	1845
GetPolicy .....	1847
GetProvisionedConcurrencyConfig .....	1849
Invoke .....	1850
ListFunctions .....	1864
ListProvisionedConcurrencyConfigs .....	1875
ListTags .....	1877
ListVersionsByFunction .....	1878
PublishVersion .....	1882
PutFunctionConcurrency .....	1883
PutProvisionedConcurrencyConfig .....	1884
RemovePermission .....	1885
TagResource .....	1887
UntagResource .....	1888
UpdateAlias .....	1889
UpdateFunctionCode .....	1890
UpdateFunctionConfiguration .....	1903
Escenarios .....	1913
Confirmación de manera automática a los usuarios conocidos con una función de Lambda .....	1914
Migración en forma automática los usuarios conocidos con una función de Lambda .....	1933
Primeros pasos con las funciones .....	1955

Escriba datos de actividad personalizados con una función de Lambda tras la autenticación de usuario de Amazon Cognito .....	2069
Ejemplos sin servidor .....	2089
Conexión a una base de datos de Amazon RDS en una función de Lambda .....	2090
Invocar una función de Lambda desde un desencadenador de Kinesis .....	2107
Invocación de una función de Lambda desde un desencadenador de DynamoDB .....	2117
Invocación de una función de Lambda desde un desencadenador de Amazon DocumentDB .....	2127
Invocación de una función de Lambda desde un desencadenador de Amazon MSK .....	2135
Invocación de una función de Lambda desde un desencadenador de Amazon S3 .....	2137
Invocar una función de Lambda desde un desencadenador de Amazon SNS .....	2148
Invocar una función de Lambda desde un desencadenador de Amazon SQS .....	2158
Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis .....	2167
Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB .....	2180
Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS. ....	2192
Ejemplos de servicios cruzados .....	2202
Creación de una API REST para realizar un seguimiento de datos de COVID-19 .....	2202
Crear una biblioteca de préstamos de API de REST .....	2203
Creación de una aplicación de mensajería .....	2204
Crear una aplicación sin servidor para administrar fotos .....	2205
Creación una aplicación de chat de websocket .....	2210
Creación de una aplicación para analizar los comentarios de los clientes .....	2211
Invocación de una función de Lambda desde un navegador .....	2217
Transformación de datos con S3 Object Lambda .....	2218
Uso de API Gateway para invocar una función de Lambda .....	2218
Usar Step Functions para invocar funciones de Lambda .....	2220
Usar eventos programados para invocar una función de Lambda .....	2222
Cuotas de Lambda .....	2225
Informática y almacenamiento .....	2225
Configuración, implementación y ejecución de funciones .....	2226
Solicitudes de la API de Lambda .....	2228
Otros servicios .....	2230
Historial de documentos .....	2231



---

Actualizaciones anteriores ..... 2260

# ¿Qué es AWS Lambda?

Puede utilizar AWS Lambda puede ejecutar código sin aprovisionar ni administrar servidores.

Lambda ejecuta el código en una infraestructura de computación de alta disponibilidad y realiza todas las tareas de administración de los recursos de computación, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como las funciones de registro. Con Lambda, lo único que tiene que hacer es suministrar el código en uno de los tiempos de ejecución de lenguaje compatibles con Lambda.

Organice su código en Funciones de Lambda. El servicio de Lambda ejecuta la función solo cuando es necesario y escala automáticamente. Solo pagará por el tiempo informático que consuma; no se aplican cargos cuando el código no se está ejecutando. Para más información, consulte [Precios de AWS Lambda](#).

## Tip

Para obtener información sobre cómo crear soluciones sin servidor, consulte la [Guía para desarrolladores sin servidor](#).

## Cuándo debe utilizar Lambda

Lambda es un servicio informático ideal para situaciones de aplicaciones que necesitan escalar verticalmente de forma rápida y reducir verticalmente a cero cuando no hay demanda. Por ejemplo, puede utilizar Lambda para los siguientes procesos:

- **Procesamiento de archivos:** utilice Amazon Simple Storage Service (Amazon S3) para iniciar el procesamiento de datos de Lambda en tiempo real después de la carga.
- **Procesamiento de flujos:** utilice Lambda y Amazon Kinesis para procesar datos de streaming en tiempo real en el seguimiento de la actividad de las aplicaciones, el procesamiento de pedidos de transacciones, el análisis de secuencias de clics, la limpieza de datos, el filtrado de registros, la indexación, el análisis de las redes sociales, la telemetría de datos de dispositivos de Internet de las cosas (IoT) y la medición de valores.
- **Aplicaciones web:** combine Lambda con otros servicios de AWS para crear aplicaciones web potentes que escalen verticalmente de manera automática y se ejecuten en una configuración de alta disponibilidad en varios centros de datos.

- Backends de IoT: cree backends sin servidor con Lambda para gestionar las solicitudes de API de Web, dispositivos móviles, IoT y terceros.
- Backends móviles: cree backends con Lambda y Amazon API Gateway para autenticar y procesar las solicitudes de API. Utilice AWS Amplify para integrar fácilmente a sus frontends de iOS, Android, Web y React Native.

Cuando se utiliza Lambda, solo es necesario preocuparse por el código. Lambda administra la flota de computación, que ofrece una combinación equilibrada de memoria, CPU, red y otros recursos para ejecutar su código. Como Lambda administra estos recursos, no puede iniciar sesión en instancias de informática ni personalizar el sistema operativo entiempos de ejecución proporcionados. Lambda realiza actividades operacionales y administrativas en su nombre, incluida la administración de la capacidad, la supervisión y el registro de las funciones de Lambda.

## Características principales

Las siguientes características clave lo ayudan a desarrollar aplicaciones escalables de Lambda, seguras y fácilmente extensibles:

### [Variables de entorno](#)

Utilice variables de entorno para ajustar el comportamiento de la función sin actualizar el código.

### [Versiones](#)

Administre la implementación de las funciones con versiones, de modo que, por ejemplo, una nueva función pueda utilizarse para realizar pruebas beta sin que esto afecte a los usuarios de la versión de producción estable.

### [Imágenes de contenedor](#)

Cree una imagen de contenedor para una función de Lambda mediante una imagen base proporcionada por AWS o una imagen base alternativa, de modo que pueda reutilizar las herramientas de contenedor existentes o implementar cargas de trabajo más grandes basadas en dependencias de tamaño modificable, como el machine learning.

### [Capas](#)

Comprima bibliotecas y otras dependencias para reducir el tamaño de los archivos de implementación y acelerar la implementación del código.

## [Extensiones de Lambda](#)

Aumente las funciones de Lambda con herramientas de supervisión, observabilidad, seguridad y gobernanza.

### [URL de funciones](#)

Agregue un punto de conexión HTTP(S) dedicado a la función de Lambda.

### [Transmisión de respuestas](#)

Configure las URL de su función de Lambda para transmitir las cargas de respuesta a los clientes desde funciones de Node.js, mejorar el rendimiento del tiempo hasta el primer byte (TTFB) o devolver cargas más grandes.

### [Controles de concurrencia y escala](#)

Aplique un control detallado sobre el escalado y la capacidad de respuesta de las aplicaciones de producción.

### [Firma de código](#)

Compruebe que solo los desarrolladores aprobados publiquen código de confianza e inalterado en sus funciones de Lambda

### [Red privada](#)

Cree una red privada para recursos como bases de datos, instancias de caché o servicios internos.

### [Acceso al sistema de archivos](#)

Configure una función para montar Amazon Elastic File System (Amazon EFS) en un directorio local, de modo que el código de función pueda acceder y modificar los recursos compartidos de forma segura y en alta concurrencia.

### [Lambda SnapStart para Java](#)

Mejore hasta 10 veces el rendimiento de inicio de los tiempos de ejecución de Java sin costo adicional, normalmente sin cambios en el código de función.

# Creación de su primera función de Lambda

Para empezar a utilizar Lambda, utilice la consola de Lambda para crear una función. En unos minutos, puede crear e implementar una función y probarla en la consola.

A medida que realice el tutorial, aprenderá algunos conceptos fundamentales de Lambda, por ejemplo, cómo pasar argumentos a su función mediante el objeto de evento de Lambda. También aprenderá a devolver los resultados de registros de su función y a ver los registros de invocación de la función en Registros de CloudWatch.

Para simplificar, cree su función mediante el tiempo de ejecución de Python o Node.js. Con estos lenguajes interpretados, puede editar el código de función directamente en el editor de código integrado en la consola. Con lenguajes compilados como Java y C#, debe crear un paquete de implementación en su máquina de compilación local y cargarlo en Lambda. Para obtener información sobre cómo implementar funciones en Lambda mediante otros tiempos de ejecución, consulte los enlaces de la sección de [the section called “Recursos adicionales y próximos pasos”](#).

## Tip

Para obtener información sobre cómo crear soluciones sin servidor, consulte la [Guía para desarrolladores sin servidor](#).

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

#### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica

recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

## Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

### Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

## Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

## Cree una función de Lambda con la consola.

En este ejemplo, la función toma un objeto de JSON que contiene dos valores enteros etiquetados como "length" y "width". La función multiplica estos valores para calcular un área y los devuelve como una cadena de JSON.

La función también imprime el área calculada, junto con el nombre de su grupo de registro de CloudWatch. Más adelante en el tutorial, aprenderá a utilizar [Registros de CloudWatch](#) para ver los registros de la invocación de sus funciones.

Para crear la función, primero debe utilizar la consola a fin de crear una función "Hola, mundo" básica. En el siguiente paso, deberá agregar su propio código de función.

Para crear una función de Lambda "Hola, mundo" con la consola, realice lo siguiente:

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija Crear función.

3. Seleccione Crear desde cero.
4. En el panel de Información básica, ingrese **myLambdaFunction** para el Nombre de la función.
5. Para Tiempo de ejecución, elija Node.js 20.x o Python 3.12
6. Establezca la arquitectura en x86\_64 y elija Crear función.

Lambda crea una función que devuelve el mensaje `Hello from Lambda!`. Lambda también crea un rol de ejecución para su función. Un [rol de ejecución](#) es un rol de AWS Identity and Access Management (IAM) que concede a la función de Lambda permiso para acceder a recursos y Servicios de AWS. En el caso de su función, el rol que crea Lambda otorga permisos básicos para escribir en Registros de CloudWatch.

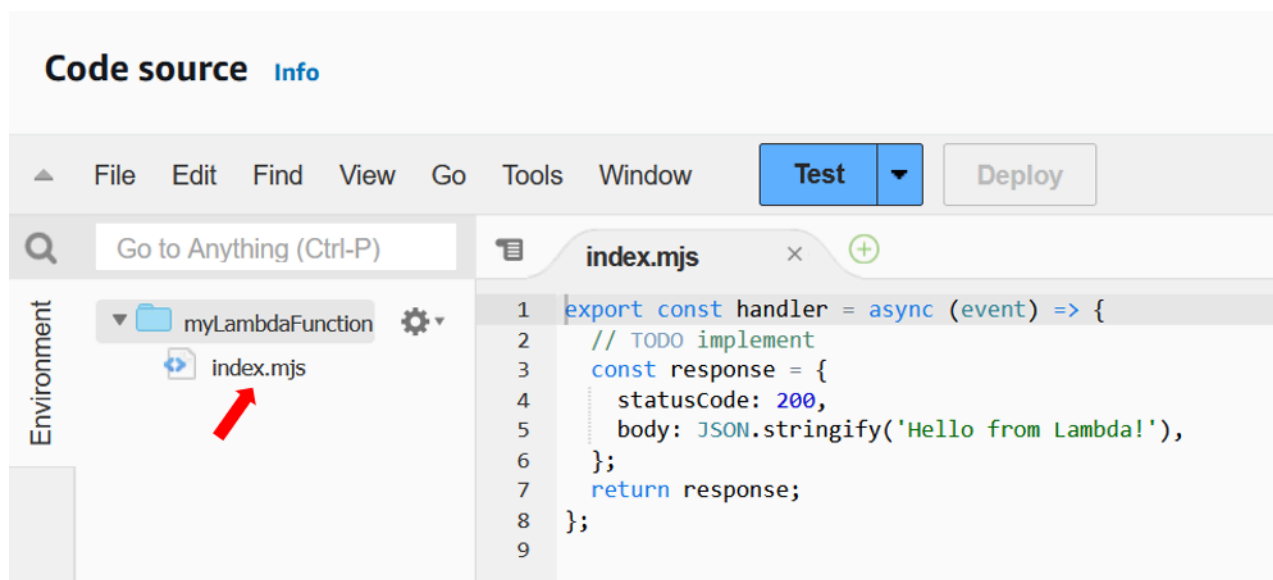
Ahora puede utilizar el editor de código integrado en la consola para reemplazar el código “Hola, mundo” que Lambda creó por su propio código de función.

## Node.js

### Modificación del código en la consola

1. Elija la pestaña Código.

En el editor de código integrado de la consola, debería ver el código de función que creó Lambda. Si no ve la pestaña `index.mjs` en el editor de código, seleccione `index.mjs` en el explorador de archivos, como se muestra en el siguiente diagrama.



2. Pegue el siguiente código en la pestaña `index.mjs`, que reemplaza el código que creó Lambda.



```
export const handler = async (event, context) => {

  const length = event.length;
  const width = event.width;
  let area = calculateArea(length, width);
  console.log(`The area is ${area}`);

  console.log('CloudWatch log group: ', context.logGroupName);

  let data = {
    "area": area,
  };
  return JSON.stringify(data);

  function calculateArea(length, width) {
    return length * width;
  }
};
```

3. Seleccione Implementar para actualizar el código de la función. Cuando Lambda haya implementado los cambios, la consola mostrará un aviso en el que se le indicará que la función se ha actualizado de forma correcta.

### Comprensión del código de función

Antes de pasar al siguiente paso, dediquemos un momento a analizar el código de función y comprender algunos conceptos clave de Lambda.

- El controlador de Lambda:

La función de Lambda contiene una función de Node.js denominada `handler`. Una función de Lambda en Node.js puede contener más de una función de Node.js, pero la función de controlador siempre es el punto de entrada a su código. Cuando se invoca la función, Lambda ejecuta este método.

Cuando creó la función “Hola, mundo” mediante la consola, Lambda estableció de forma automática el nombre del método controlador de la función en `handler`. Asegúrese de no editar el nombre de esta función de Node.js. Si lo hace, Lambda no podrá ejecutar el código cuando invoque la función.

Para obtener más información sobre el controlador de Lambda en Node.js, consulte [the section called “Controlador”](#).

- El objeto de evento de Lambda:

La función `handler` toma dos argumentos, `event` y `context`. Un evento en Lambda es un documento con formato JSON que contiene datos para que una función los procese.

Si otro Servicio de AWS invoca su función, el objeto de evento contiene información sobre el evento que provocó la invocación. Por ejemplo, si un bucket de Amazon Simple Storage Service (Amazon S3) invoca su función al cargar un objeto, el evento contendrá el nombre del bucket de Amazon S3 y la clave del objeto.

En este ejemplo, creará un evento en la consola al ingresar un documento con formato JSON con dos pares clave-valor.

- El objeto de contexto de Lambda:

El segundo argumento que utiliza su función es `context`. Lambda pasa el objeto de contexto a su función de forma automática. El objeto de contexto contiene información sobre la invocación de la función y el entorno de ejecución.

Puede utilizar el objeto de contexto para generar información sobre la invocación de su función con fines de supervisión. En este ejemplo, la función utiliza el parámetro `logGroupName` para generar el nombre de su grupo de registro de CloudWatch.

Para obtener más información sobre el objeto de contexto de Lambda en Node.js, consulte [the section called “Context”](#).

- Registro en Lambda:

Con Node.js, puede utilizar métodos consola como `console.log` y `console.error` para enviar información al registro de su función. El código de ejemplo utiliza instrucciones `console.log` para generar el área calculada y el nombre del grupo de Registros de CloudWatch de la función. También puede utilizar cualquier biblioteca de registro que escriba en `stdout` o `stderr`.

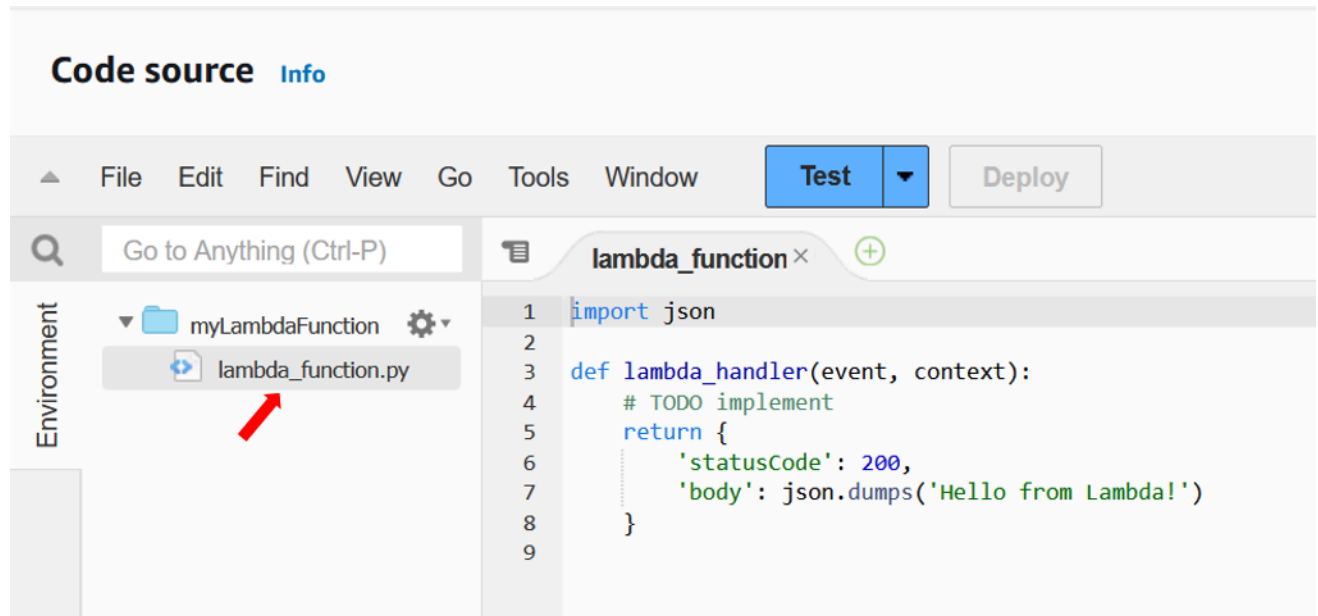
Para obtener más información, consulte [the section called “Registro”](#). Para obtener información sobre los registros en otros tiempos de ejecución, consulte las páginas “Construir con” a fin de conocer los tiempos de ejecución que le interesen.

## Python

### Modificación del código en la consola

1. Elija la pestaña Código.

En el editor de código integrado de la consola, debería ver el código de función que creó Lambda. Si no ve la pestaña `lambda_function.py` en el editor de código, seleccione `lambda_function.py` en el explorador de archivos, como se muestra en el siguiente diagrama.



2. Pegue el siguiente código en la pestaña `lambda_function.py`, que reemplaza el código que creó Lambda.

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']
```

```
area = calculate_area(length, width)
print(f"The area is {area}")

logger.info(f"CloudWatch logs group: {context.log_group_name}")

# return the calculated area as a JSON string
data = {"area": area}
return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

3. Seleccione Implementar para actualizar el código de la función. Cuando Lambda haya implementado los cambios, la consola mostrará un aviso en el que se le indicará que la función se ha actualizado de forma correcta.

## Comprensión del código de función

Antes de pasar al siguiente paso, dediquemos un momento a analizar el código de función y comprender algunos conceptos clave de Lambda.

- El controlador de Lambda:

La función de Lambda contiene una función de Python denominada `lambda_handler`. Una función de Lambda en Python puede contener más de una función de Python, pero la función de controlador siempre es el punto de entrada a su código. Cuando se invoca la función, Lambda ejecuta este método.

Cuando creó la función “Hola, mundo” mediante la consola, Lambda estableció de forma automática el nombre del método controlador de la función en `lambda_handler`. Asegúrese de no editar el nombre de esta función de Python. Si lo hace, Lambda no podrá ejecutar el código cuando invoque la función.

Para obtener más información sobre el controlador de Lambda en Python, consulte [the section called “Controlador”](#).

- El objeto de evento de Lambda:

La función `lambda_handler` toma dos argumentos, `event` y `context`. Un evento en Lambda es un documento con formato JSON que contiene datos para que una función los procese.

Si otro Servicio de AWS invoca su función, el objeto de evento contiene información sobre el evento que provocó la invocación. Por ejemplo, si un bucket de Amazon Simple Storage Service (Amazon S3) invoca su función al cargar un objeto, el evento contendrá el nombre del bucket de Amazon S3 y la clave del objeto.

En este ejemplo, creará un evento en la consola al ingresar un documento con formato JSON con dos pares clave-valor.

- El objeto de contexto de Lambda:

El segundo argumento que utiliza su función es `context`. Lambda pasa el objeto de contexto a su función de forma automática. El objeto de contexto contiene información sobre la invocación de la función y el entorno de ejecución.

Puede utilizar el objeto de contexto para generar información sobre la invocación de su función con fines de supervisión. En este ejemplo, la función utiliza el parámetro `log_group_name` para generar el nombre de su grupo de registro de CloudWatch.

Para obtener más información sobre el objeto de contexto de Lambda en Python, consulte [the section called “Context”](#).

- Registro en Lambda:

Con Python, puede utilizar una instrucción `print` o una biblioteca de registro de Python para enviar información al registro de la función. Para ilustrar la diferencia en lo que se captura, el código de ejemplo utiliza ambos métodos. En una aplicación de producción, se recomienda utilizar una biblioteca de registro.

Para obtener más información, consulte [the section called “Registro y supervisión de las funciones de Lambda de Python”](#). Para obtener información sobre los registros en otros tiempos de ejecución, consulte las páginas “Construir con” a fin de conocer los tiempos de ejecución que le interesen.

## Invocar la función de Lambda mediante la consola

Para invocar la función mediante la consola de Lambda, primero debe crear un evento de prueba a fin de enviarlo a la función. El evento es un documento con formato JSON que contiene dos pares clave-valor con las claves `"length"` y `"width"`.

Para crear el evento de prueba, realice lo siguiente:

1. En el panel Código fuente, elija Probar.
2. Seleccione Crear un evento nuevo.
3. En Nombre del evento, escriba **myTestEvent**.
4. En el panel de Evento JSON, sustituya los valores predeterminados al pegar lo siguiente:

```
{
  "length": 6,
  "width": 7
}
```

5. Seleccione Guardar.

Ahora puede probar la función y utilizar la consola de Lambda y Registros de CloudWatch para ver los registros de la invocación de la función.

Prueba de la función y visualización de los registros de invocación en la consola

- En el panel Código fuente, elija Probar. Cuando la función termine de ejecutarse, verá los registros de respuesta y función en la pestaña Resultados de la ejecución. Debería ver resultados similares a los siguientes.

Node.js

```
Test Event Name
myTestEvent

Response
"{\"area\":42}"

Function Logs
START RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Version: $LATEST
2023-08-31T23:39:45.313Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO The area is
42
2023-08-31T23:39:45.331Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO CloudWatch
log group: /aws/lambda/myLambdaFunction
END RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a
REPORT RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Duration: 20.67 ms Billed
Duration: 21 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration:
163.87 ms
```

```
Request ID
5c012b0a-18f7-4805-b2f6-40912935034a
```

## Python

```
Test Event Name
myTestEvent

Response
"{\"area\": 42}"

Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO] 2023-08-31T23:43:26.428Z 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b CloudWatch
logs group: /aws/lambda/myLambdaFunction
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Duration: 1.42 ms Billed
Duration: 2 ms Memory Size: 128 MB Max Memory Used: 39 MB Init Duration: 123.74
ms

Request ID
2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
```

En este ejemplo, ha invocado el código mediante la característica de prueba de la consola. Esto significa que puede ver los resultados de ejecución de la función directamente en la consola. Cuando la función se invoca fuera de la consola, debe utilizar Registros de CloudWatch.

### Visualización de los registros de invocación de la función en Registros de CloudWatch

1. En la consola de CloudWatch, abra la página [Log groups \(grupos de registro\)](#).
2. Elija el grupo de registro para la función (`/aws/lambda/myLambdaFunction`). Es el nombre del grupo de registro que su función imprimió en la consola.
3. En la pestaña Flujos de registro, elija el flujo de registro para la invocación de la función.

Debería ver una salida similar a esta:

## Node.js

```
INIT_START Runtime Version: nodejs:20.v13    Runtime Version ARN:
arn:aws:lambda:us-
west-2::runtime:e3aaabf6b92ef8755eaae2f4bfdcb7eb8c4536a5e044900570a42bdba7b869d9
START RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20 Version: $LATEST
2023-08-23T22:04:15.809Z    5c012b0a-18f7-4805-b2f6-40912935034a    INFO The area
is 42
2023-08-23T22:04:15.810Z    aba6c0fc-cf99-49d7-a77d-26d805dacd20    INFO
CloudWatch log group: /aws/lambda/myLambdaFunction
END RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20
REPORT RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20    Duration: 17.77 ms
Billed Duration: 18 ms    Memory Size: 128 MB    Max Memory Used: 67 MB    Init
Duration: 178.85 ms
```

## Python

```
INIT_START Runtime Version: python:3.12.v16    Runtime Version ARN:
arn:aws:lambda:us-
west-2::runtime:ca202755c87b9ec2b58856efb7374b4f7b655a0ea3deb1d5acc9aee9e297b072
START RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e Version: $LATEST
The area is 42
[INFO] 2023-09-01T00:05:22.464Z 9315ab6b-354a-486e-884a-2fb2972b7d84 CloudWatch
logs group: /aws/lambda/myLambdaFunction
END RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e
REPORT RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e    Duration: 1.15 ms
Billed Duration: 2 ms    Memory Size: 128 MB    Max Memory Used: 40 MB
```

## Limpieza

Cuando haya terminado de trabajar con la función de ejemplo, elimínela. También puede eliminar el [rol de ejecución](#) creado por la consola y el grupo de registro que almacena los registros de la función.

Para eliminar una función de Lambda

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija una función.
3. Elija Actions (Acciones), Delete (Eliminar).



4. En el cuadro de diálogo Delete function (Eliminar función), escriba delete y, a continuación, elija Delete (Eliminar).

Para eliminar el grupo de registros

1. En la consola de CloudWatch, abra la página [Log groups \(Grupos de registro\)](#).
2. Seleccione el grupo de registros de la función (/aws/lambda/my-function).
3. Elija Acciones, Eliminar grupo(s) de registro(s).
4. En el cuadro de diálogo Delete log group(s), Eliminar grupo(s) de registro(s) elija Delete (Eliminar).

Cómo eliminar el rol de ejecución

1. Abra la página de [Roles](#) de la consola de AWS Identity and Access Management (IAM).
2. Seleccione el rol de ejecución de la función, por ejemplo, myLambdaFunction-role-*31exxmpl*.
3. Elija Eliminar.
4. En el cuadro de diálogo Eliminar rol, escriba el nombre del rol y, a continuación, elija Eliminar.

Puede automatizar la creación y limpieza de funciones, grupos de registros y roles con AWS CloudFormation y AWS Command Line Interface (AWS CLI).

## Recursos adicionales y próximos pasos

Ahora que ha creado y probado una función de Lambda simple con la consola, siga estos pasos:

- Aprenda a agregar dependencias a su código y a implementarlo mediante un paquete de implementación en formato .zip. Elija entre los siguientes enlaces los lenguajes que le interesen.

Node.js

Consulte [the section called “Implementar archivos de archivos .zip”](#)

Typescript

Consulte [the section called “Implementar archivos de archivos .zip”](#)

## Python

Consulte [the section called “Implementar archivos de archivos .zip”](#)

## Ruby

Consulte [the section called “Implementar archivos de archivos .zip”](#)

## Java

Consulte [the section called “Implementar archivos de archivos .zip”](#)

## Go

Consulte [the section called “Implementar archivos de archivos .zip”](#)

## C#

Consulte [the section called “Paquete de implementación”](#).

- Realice el tutorial [Uso de un desencadenador de Amazon S3 para invocar una función de Lambda](#) a fin de aprender a configurar una función de Lambda para que la invoque otro Servicio de AWS.
- Elija uno de los siguientes tutoriales para ver un ejemplo más complejo del uso de Lambda con otros Servicios de AWS.
  - [Uso de Lambda con API Gateway](#): cree una API de REST de Amazon API Gateway que invoque una función de Lambda.
  - [Uso de una función de Lambda para acceder a una base de datos de Amazon RDS](#): utilice una función de Lambda para escribir datos en una base de datos de Amazon Relational Database Service (Amazon RDS) a través de RDS Proxy.
  - [Uso de un desencadenador de Amazon S3 para crear imágenes en miniatura](#): utilice una función de Lambda para crear una miniatura cada vez que se cargue un archivo de imagen en un bucket de Amazon S3.

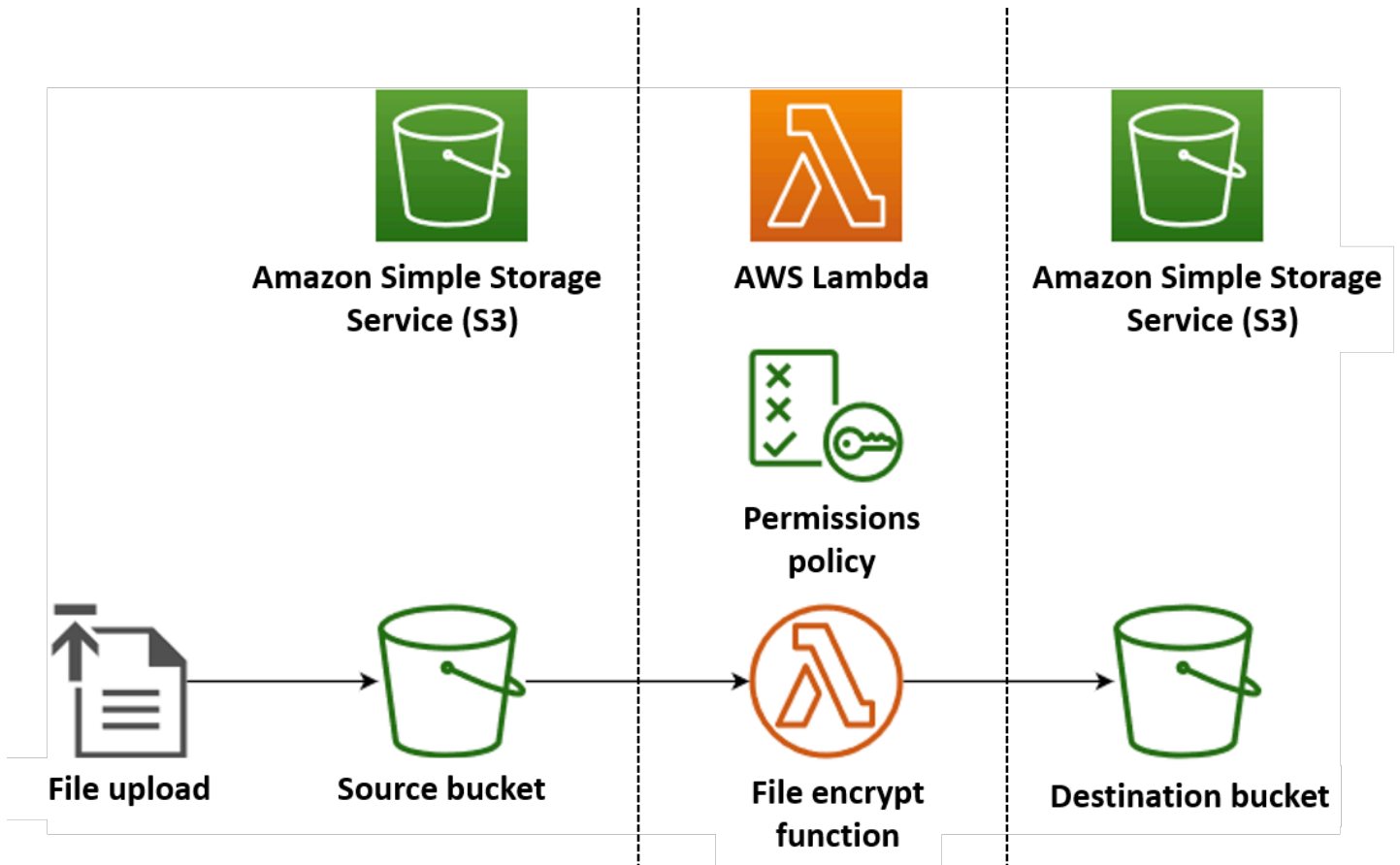
# Creación de una aplicación de procesamiento de archivos sin servidor

Uno de los casos de uso más comunes de Lambda es llevar a cabo tareas de procesamiento de archivos. Por ejemplo, puede utilizar una función de Lambda para crear archivos PDF de forma automática a partir de archivos HTML o imágenes, o para crear miniaturas cuando un usuario carga una imagen.

En este ejemplo, crea una aplicación que cifra los archivos PDF de forma automática cuando se cargan en un bucket de Amazon Simple Storage Service (Amazon S3). Para implementar esta aplicación, necesita crear los siguientes recursos:

- Un bucket de S3 para que los usuarios puedan cargar los archivos PDF
- Una función de Lambda en Python que lea el archivo cargado y cree una versión cifrada y protegida por contraseña
- Un segundo bucket de S3 en el que Lambda pueda guardar el archivo cifrado

También debe crear una política de AWS Identity and Access Management (IAM) para conceder permiso a la función de Lambda para llevar a cabo operaciones de lectura y escritura en los buckets de S3.

**Tip**

Si es la primera vez que utiliza Lambda, recomendamos que complete el tutorial [Creación de su primera función](#) antes de crear esta aplicación de ejemplo.

Para implementar la aplicación de forma manual, cree y configure los recursos mediante la AWS Management Console o con la AWS Command Line Interface (AWS CLI). También puede implementar la aplicación mediante AWS Serverless Application Model (AWS SAM). AWS SAM es una herramienta de Infraestructura como código (IaC). Con la IaC no se crean recursos de forma manual, sino que se definen en código y, a continuación, se implementan automáticamente.

Si desea obtener más información sobre el uso de Lambda con la IaC antes de implementar esta aplicación de ejemplo, consulte [Infraestructura como código \(IaC\)](#).

## Requisitos previos

Antes de poder crear la aplicación de ejemplo, asegúrese de tener instaladas las herramientas de línea de comando necesarias.

- AWS CLI

Puede implementar los recursos de la aplicación de forma manual mediante la AWS Management Console o la AWS CLI. Para usar la CLI, debe instalarla siguiendo las [instrucciones de instalación](#) en la Guía del usuario de AWS Command Line Interface.

- CLI de AWS SAM

Si desea implementar la aplicación de ejemplo mediante AWS SAM, debe instalar tanto la AWS CLI como la CLI de AWS SAM. Para instalar la CLI de AWS SAM, siga las [instrucciones de instalación](#) en la Guía del usuario de AWS SAM.

- módulo pytest

Una vez que haya implementado la aplicación, puede probarla con un script de prueba de Python automatizado que proporcionamos. Para usar este script, instale el paquete `pytest` en su entorno de desarrollo local ejecutando el siguiente comando:

```
pip install pytest
```

Para implementar la aplicación con AWS SAM, [Docker](#) también debe estar instalado en la máquina de compilación.

## Descarga de los archivos de la aplicación de ejemplo

Para crear y probar la aplicación de ejemplo, debe crear los siguientes archivos en el directorio del proyecto:

- `lambda_function.py`: es el código de la función Python para la función de Lambda que lleva a cabo el cifrado de archivos
- `requirements.txt`: es un archivo de manifiesto que define las dependencias que requiere el código de la función de Python
- `template.yaml`: es una plantilla de AWS SAM que puede usar para implementar la aplicación

- `test_pdf_encrypt.py`: es un script de prueba que puede usar para probar la aplicación de forma automática
- `pytest.ini`: es un archivo de configuración para el script de prueba

Expanda las siguientes secciones para ver el código y obtener más información sobre la función de cada archivo a la hora de crear y probar la aplicación. Para crear los archivos en la máquina local, copie y pegue el siguiente código o descargue los archivos del [repositorio `aws-lambda-developer-guide` de GitHub](#).

## Código de la función de Python

Copie y pegue el siguiente código en un archivo denominado `lambda_function.py`.

```
from pypdf import PdfReader, PdfWriter
import uuid
import os
from urllib.parse import unquote_plus
import boto3

# Create the S3 client to download and upload objects from S3
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Iterate over the S3 event object and get the key for all uploaded files
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key']) # Decode the S3 object key to
        # remove any URL-encoded characters
        download_path = f'/tmp/{uuid.uuid4()}.pdf' # Create a path in the Lambda tmp
        # directory to save the file to
        upload_path = f'/tmp/converted-{uuid.uuid4()}.pdf' # Create another path to
        # save the encrypted file to

        # If the file is a PDF, encrypt it and upload it to the destination S3 bucket
        if key.lower().endswith('.pdf'):
            s3_client.download_file(bucket, key, download_path)
            encrypt_pdf(download_path, upload_path)
            encrypted_key = add_encrypted_suffix(key)
            s3_client.upload_file(upload_path, f'{bucket}-encrypted', encrypted_key)

# Define the function to encrypt the PDF file with a password
def encrypt_pdf(file_path, encrypted_file_path):
```

```
reader = PdfReader(file_path)
writer = PdfWriter()

for page in reader.pages:
    writer.add_page(page)

# Add a password to the new PDF
writer.encrypt("my-secret-password")

# Save the new PDF to a file
with open(encrypted_file_path, "wb") as file:
    writer.write(file)

# Define a function to add a suffix to the original filename after encryption
def add_encrypted_suffix(original_key):
    filename, extension = original_key.rsplit('.', 1)
    return f'{filename}_encrypted.{extension}'
```

#### Note

En este código de ejemplo, la contraseña del archivo cifrado (`my-secret-password`) está codificada en el código de la función. En aplicaciones de producción, no incluya información confidencial como contraseñas en el código de la función. Utilice AWS Secrets Manager para almacenar los parámetros confidenciales de forma segura.

El código de la función de Python contiene tres funciones: la [función de controlador](#) que Lambda ejecuta cuando se invoca la función y dos funciones independientes denominadas `add_encrypted_suffix` y `encrypt_pdf` a las que el controlador llama para que lleve a cabo el cifrado del PDF.

Cuando Amazon S3 invoca la función, Lambda pasa un argumento de evento con formato JSON a la función que contiene detalles sobre el evento que provocó la invocación. En este caso, la información incluye el nombre del bucket de S3 y las claves de objeto de los archivos cargados. Para obtener más información sobre el formato del objeto de evento de Amazon S3, consulte [the section called "S3"](#).

A continuación, la función utiliza AWS SDK for Python (Boto3) para descargar los archivos PDF especificados en el objeto de evento a su directorio de almacenamiento temporal local, antes de cifrarlos mediante la biblioteca [pypdf](#).

Por último, la función utiliza el SDK de Boto3 para almacenar el archivo cifrado en el bucket de destino de S3.

## archivo de manifiesto **requirements.txt**

Copie y pegue el siguiente código en un archivo denominado `requirements.txt`.

```
boto3
pypdf
```

En este ejemplo, el código de la función solo tiene dos dependencias que no forman parte de la biblioteca estándar de Python: el SDK para Python (Boto3) y el paquete `pypdf` que la función utiliza para cifrar el PDF.

### Note

Como parte del tiempo de ejecución de Lambda, se incluye una versión del SDK para Python (Boto3), de modo que el código se ejecute sin agregar Boto3 al paquete de implementación de la función. Sin embargo, para mantener el control total de las dependencias de la función y evitar posibles problemas de alineación de versiones, la práctica recomendada para Python es incluir todas las dependencias de la función en el paquete de implementación de la función. Consulte [the section called “Dependencias de tiempo de ejecución en Python”](#) para obtener más información.

## Plantilla de AWS SAM

Copie y pegue el siguiente código en un archivo denominado `template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  EncryptPDFFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: EncryptPDF
      Architectures: [x86_64]
      CodeUri: ./
```



```
Handler: lambda_function.lambda_handler
Runtime: python3.12
Timeout: 15
MemorySize: 256
LoggingConfig:
  LogFormat: JSON
Policies:
  - AmazonS3FullAccess
Events:
  S3Event:
    Type: S3
    Properties:
      Bucket: !Ref PDFSourceBucket
      Events: s3:ObjectCreated:*
```

```
PDFSourceBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: EXAMPLE-BUCKET
```

```
EncryptedPDFBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: EXAMPLE-BUCKET-encrypted
```

La plantilla AWS SAM define los recursos que crea para la aplicación. En este ejemplo, la plantilla define una función de Lambda con el tipo `AWS::Serverless::Function` y dos buckets de S3 con el tipo `AWS::S3::Bucket`. Los nombres de los buckets especificados en la plantilla son marcadores de posición. Antes de implementar la aplicación mediante AWS SAM, debe editar la plantilla para cambiar el nombre de los buckets por nombres únicos a nivel mundial que cumplan con las [reglas de nomenclatura para los buckets de S3](#). Este paso se explica con más detalle en [the section called “Implementación de los recursos mediante AWS SAM”](#).

La definición del recurso de la función de Lambda configura un desencadenador para la función mediante la propiedad del evento `S3Event`. Este desencadenador hace que se invoque la función cada vez que se crea un objeto en el bucket de origen.

La definición de la función también especifica una política de AWS Identity and Access Management (IAM) que se adjuntará al [rol de ejecución](#) de la función. La [política administrada de AWS AmazonS3FullAccess](#) proporciona a su función los permisos que necesita para leer y escribir objetos en Amazon S3.

## Script de prueba automatizado

Copie y pegue el siguiente código en un archivo denominado `test_pdf_encrypt.py`.

```
import boto3
import json
import pytest
import time
import os

@pytest.fixture
def lambda_client():
    return boto3.client('lambda')

@pytest.fixture
def s3_client():
    return boto3.client('s3')

@pytest.fixture
def logs_client():
    return boto3.client('logs')

@pytest.fixture(scope='session')
def cleanup():
    # Create a new S3 client for cleanup
    s3_client = boto3.client('s3')

    yield

    # Cleanup code will be executed after all tests have finished

    # Delete test.pdf from the source bucket
    source_bucket = 'EXAMPLE-BUCKET'
    source_file_key = 'test.pdf'
    s3_client.delete_object(Bucket=source_bucket, Key=source_file_key)
    print(f"\nDeleted {source_file_key} from {source_bucket}")

    # Delete test_encrypted.pdf from the destination bucket
    destination_bucket = 'EXAMPLE-BUCKET-encrypted'
    destination_file_key = 'test_encrypted.pdf'
    s3_client.delete_object(Bucket=destination_bucket, Key=destination_file_key)
    print(f"Deleted {destination_file_key} from {destination_bucket}")

@pytest.mark.order(1)
```

```
def test_source_bucket_available(s3_client):
    s3_bucket_name = 'EXAMPLE-BUCKET'
    file_name = 'test.pdf'
    file_path = os.path.join(os.path.dirname(__file__), file_name)

    file_uploaded = False
    try:
        s3_client.upload_file(file_path, s3_bucket_name, file_name)
        file_uploaded = True
    except:
        print("Error: couldn't upload file")

    assert file_uploaded, "Could not upload file to S3 bucket"

@pytest.mark.order(2)
def test_lambda_invoked(logs_client):

    # Wait for a few seconds to make sure the logs are available
    time.sleep(5)

    # Get the latest log stream for the specified log group
    log_streams = logs_client.describe_log_streams(
        logGroupName='/aws/lambda/EncryptPDF',
        orderBy='LastEventTime',
        descending=True,
        limit=1
    )

    latest_log_stream_name = log_streams['logStreams'][0]['logStreamName']

    # Retrieve the log events from the latest log stream
    log_events = logs_client.get_log_events(
        logGroupName='/aws/lambda/EncryptPDF',
        logStreamName=latest_log_stream_name
    )

    success_found = False
    for event in log_events['events']:
        message = json.loads(event['message'])
        status = message.get('record', {}).get('status')
        if status == 'success':
            success_found = True
```

```
        break

    assert success_found, "Lambda function execution did not report 'success' status in
logs."

@pytest.mark.order(3)
def test_encrypted_file_in_bucket(s3_client):
    # Specify the destination S3 bucket and the expected converted file key
    destination_bucket = 'EXAMPLE-BUCKET-encrypted'
    converted_file_key = 'test_encrypted.pdf'

    try:
        # Attempt to retrieve the metadata of the converted file from the destination
S3 bucket
        s3_client.head_object(Bucket=destination_bucket, Key=converted_file_key)
    except s3_client.exceptions.ClientError as e:
        # If the file is not found, the test will fail
        pytest.fail(f"Converted file '{converted_file_key}' not found in the
destination bucket: {str(e)}")

def test_cleanup(cleanup):
    # This test uses the cleanup fixture and will be executed last
    pass
```

El script de prueba automatizado ejecuta tres funciones de prueba para confirmar el correcto funcionamiento de la aplicación:

- La prueba `test_source_bucket_available` confirma que el bucket de origen se creó correctamente mediante la carga de un archivo PDF de prueba en el bucket.
- La prueba `test_lambda_invoked` consulta el último flujo de registro de los registros de CloudWatch de la función para confirmar que la función de Lambda se ejecutó correctamente cuando cargó el archivo de prueba.
- La prueba `test_encrypted_file_in_bucket` confirma que el bucket de destino contiene el archivo cifrado `test_encrypted.pdf`.

Una vez ejecutadas todas estas pruebas, el script ejecuta un paso de limpieza adicional para eliminar los archivos `test.pdf` y `test_encrypted.pdf` de los buckets de origen y destino.

Los nombres de los buckets especificados en este archivo son marcadores de posición, al igual que en la plantilla AWS SAM. Antes de ejecutar la prueba, debe editar este archivo con los nombres

reales de los buckets de la aplicación. Este paso se explica con más detalle en [the section called “Prueba de la aplicación con el script automatizado”](#)

## Prueba del archivo de configuración del script

Copie y pegue el siguiente código en un archivo denominado `pytest.ini`.

```
[pytest]
markers =
    order: specify test execution order
```

Esto es necesario para especificar el orden en el que se ejecutan las pruebas del script `test_pdf_encrypt.py`.

## Implementación de la aplicación

Puede crear e implementar los recursos para esta aplicación de ejemplo de forma manual o mediante AWS SAM. En un entorno de producción, recomendamos que utilice una herramienta de IaC como AWS SAM para implementar aplicaciones sin servidor de forma rápida y repetible sin utilizar procesos manuales.

Para este ejemplo, siga las instrucciones de la consola o la AWS CLI para aprender a configurar cada recurso de AWS por separado, o pase a [the section called “Implementación de los recursos mediante AWS SAM”](#) para implementar la aplicación de forma rápida mediante unos pocos comandos de la CLI.

## Implementación de los recursos de forma manual

Para implementar la aplicación de forma manual, lleve a cabo los siguientes pasos:

- Cree buckets de S3 de origen y destino
- Cree una función de Lambda que cifre un archivo PDF y guarde la versión cifrada de este en un bucket de S3
- Configure un desencadenador de Lambda que invoque la función cuando se carguen objetos en el bucket de origen

Siga las instrucciones en los párrafos a continuación para crear y configurar los recursos.

## Cree dos buckets de S3

Primero, cree dos buckets de S3. El primer bucket es el bucket de origen al que subirá los archivos PDF. Lambda utiliza el segundo bucket para guardar el archivo cifrado cuando se invoca la función.

### Console

Para crear buckets de S3 (consola)

1. En la consola de Amazon S3, abra la página [Buckets](#).
2. Elija Crear bucket.
3. En Configuración general, haga lo siguiente:
  - a. Para el nombre del bucket, ingrese un nombre único a nivel mundial que cumpla las [reglas de nomenclatura de bucket](#) de Amazon S3. Los nombres de bucket pueden contener únicamente letras minúsculas, números, puntos (.) y guiones (-).
  - b. Para Región de AWS, elija la [Región de AWS](#) más cercana a su ubicación geográfica. Más adelante, durante el proceso de implementación, debe crear la función de Lambda en la misma Región de AWS, por lo que debe anotar la región que eligió.
4. Deje el resto de las opciones con sus valores predeterminados y seleccione Crear bucket.
5. Repita los pasos 1 a 4 para crear el bucket de destino. En Nombre del bucket, introduzca **SOURCEBUCKET-encrypted**, donde **SOURCEBUCKET** es el nombre del bucket de origen que acaba de crear.

### AWS CLI

Para crear buckets de S3 (AWS CLI)

1. Ejecute el siguiente comando de la CLI para crear el bucket de origen. El nombre que elija para el bucket debe ser globalmente único y seguir las [reglas de nomenclatura de buckets](#) de Amazon S3. Los nombres pueden contener únicamente letras minúsculas, números, puntos (.) y guiones (-). Para `region` y `LocationConstraint`, elija la [Región de AWS](#) más cercana a su ubicación geográfica.

```
aws s3api create-bucket --bucket SOURCEBUCKET --region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2
```

Más adelante en el tutorial, debe crear la función de Lambda en la misma Región de AWS que la del bucket de origen, por lo que debe anotar la región que eligió.

2. Ejecute el siguiente comando para crear el bucket de destino. Para el nombre del bucket, debe utilizar **SOURCEBUCKET-encrypted**, donde **SOURCEBUCKET** es el nombre del bucket de origen que creó en el paso 1. Para `region` y `LocationConstraint`, elija la misma Región de AWS que usó para crear el bucket de origen.

```
aws s3api create-bucket --bucket SOURCEBUCKET-encrypted --region us-west-2 \  
--create-bucket-configuration LocationConstraint=us-west-2
```

## Crear un rol de ejecución (solo para la AWS CLI)

Un rol de ejecución es un rol de IAM que concede a la función de Lambda permiso para acceder a servicios y recursos de Servicios de AWS. Cuando crea una función con la consola de Lambda, Lambda crea un rol de ejecución de forma automática. Solo necesita crear un rol de forma manual si decide implementar la aplicación mediante la AWS CLI. Para conceder acceso de lectura y escritura a Amazon S3 a la función, debe adjuntar la [política administrada de AWS](#) `AmazonS3FullAccess`.

### Console

Este paso solo es obligatorio si elige implementar la aplicación mediante la AWS CLI.

### AWS CLI

Crear un rol de ejecución y adjuntar la política administrada **AmazonS3FullAccess** (AWS CLI)

1. Guarde el siguiente JSON en un archivo llamado `trust-policy.json`. Esta política de confianza permite a Lambda utilizar los permisos del rol al dar permiso al servicio principal `lambda.amazonaws.com` para llamar a la acción de AWS Security Token Service (AWS STS) `AssumeRole`.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "lambda.amazonaws.com"  
      },  
    },  
  ],  
}
```

```
    "Action": "sts:AssumeRole"
  }
]
}
```

- Desde el directorio en el que guardó el documento de política de confianza JSON, ejecute el siguiente comando de la CLI para crear el rol de ejecución.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
file://trust-policy.json
```

- Para adjuntar la política administrada AmazonS3FullAccess, ejecute el siguiente comando de la CLI:

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
arn:aws:iam::aws:policy/AmazonS3FullAccess
```

## Crear el paquete de despliegue de la función

Para crear una función, debe crear un paquete de despliegue que contenga el código y las dependencias de la función. Para esta aplicación, el código de la función utiliza una biblioteca independiente para el cifrado de PDF.

### Creación del paquete de implementación

- Navegue hasta el directorio del proyecto que contiene los archivos `lambda_function.py` y `requirements.txt` que creó o descargó de GitHub anteriormente y cree un nuevo directorio denominado `package`.
- Instale las dependencias especificadas en el archivo `requirements.txt` del directorio `package` ejecutando el siguiente comando:

```
pip install -r requirements.txt --target ./package/
```

- Cree un archivo `.zip` que contenga el código de la aplicación y todas sus dependencias. En Linux o macOS, ejecute los siguientes comandos desde la interfaz de la línea de comandos.

```
cd package
zip -r ../lambda_function.zip .
cd ..
```



```
zip lambda_function.zip lambda_function.py
```

En Windows, utilice la herramienta de compresión que prefiera para crear el archivo `lambda_function.zip`. Asegúrese de que el archivo `lambda_function.py` y las carpetas que contienen las dependencias estén en la raíz del archivo `.zip`.

También puede crear su paquete de implementación mediante un entorno virtual de Python.

Consulte [Uso de archivos .zip para funciones de Lambda en Python](#)

## Creación de la función de Lambda

Ahora use el paquete de implementación que creó en el paso anterior para implementar la función de Lambda.

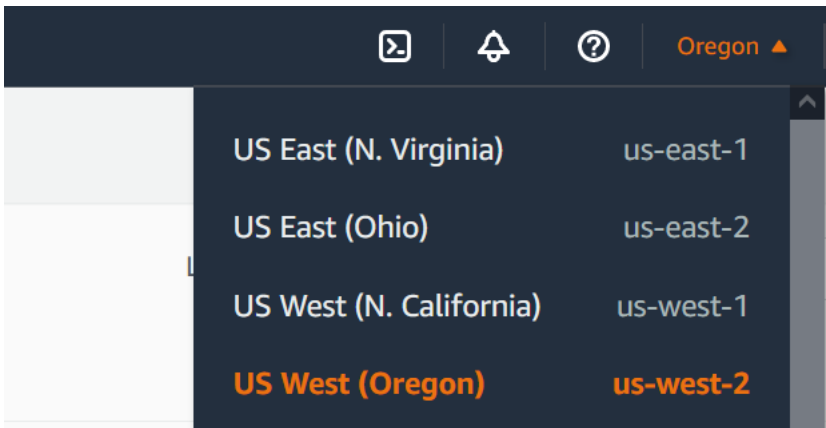
### Console

Para crear la función (consola)

Para crear una función de Lambda con la consola, primero debe crear una función básica que contenga el código “Hola, mundo”. A continuación, reemplace este código por su propio código de función mediante la carga del archivo `.zip` que creó en el paso anterior.

Para garantizar que no se agote el tiempo de espera de la función cuando cifra archivos PDF de gran tamaño, debe configurar los ajustes de memoria y tiempo de espera de la función. El formato de registro de la función también se establece como JSON. Es necesario configurar los registros con formato JSON cuando se utiliza el script de prueba proporcionado para que pueda leer el estado de invocación de la función desde los registros de CloudWatch y confirmar que la invocación se llevó a cabo de forma correcta.

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Asegúrese de trabajar en la misma Región de AWS en la que creó el bucket de S3. Puede cambiar la región por medio de la lista desplegable de la parte superior de la pantalla.



3. Elija Crear función.
4. Elija Author from scratch (Crear desde cero).
5. Bajo Basic information (Información básica), haga lo siguiente:
  - a. En Function name (Nombre de la función), introduzca **EncryptPDF**.
  - b. En Tiempo de ejecución, seleccione Python 3.12.
  - c. En Arquitectura, elija x86\_64.
6. Elija Crear función.

Para cargar el código de la función (consola)

1. En el panel Código fuente, elija Cargar desde.
2. Elija un archivo .zip.
3. Seleccione Cargar.
4. En el selector de archivos, seleccione un archivo .zip y luego elija Abrir.
5. Seleccione Guardar.

Configuración de la memoria y el tiempo de espera de la función (consola)

1. Seleccione la pestaña Configuración de la función.
2. En el panel Configuración general, seleccione Editar.
3. Establezca la Memoria en 256 MB y el Tiempo de espera en 15 segundos.
4. Seleccione Guardar.

## Configuración del formato de registro (consola)

1. Seleccione la pestaña Configuración de la función.
2. Seleccione Herramientas de supervisión y operaciones.
3. En el panel Configuración de registros, seleccione Editar.
4. Para la configuración de registro, seleccione JSON.
5. Seleccione Guardar.

## AWS CLI

Para crear la función (AWS CLI)

- Ejecute el siguiente comando desde el directorio que contiene el archivo `lambda_function.zip`. Para el parámetro `region`, reemplace `us-west-2` por la región en la que creó los buckets de S3.

```
aws lambda create-function --function-name EncryptPDF \  
--zip-file fileb://lambda_function.zip --handler lambda_function.lambda_handler \  
\  
--runtime python3.12 --timeout 15 --memory-size 256 \  
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-west-2 \  
--logging-config LogFormat=JSON
```

## Configuración de un desencadenador de Amazon S3 para invocar una función

Para que la función de Lambda se ejecute cuando carga un archivo al bucket de origen, debe configurar un desencadenador para la función. Puede configurar el desencadenador de Amazon S3 mediante la consola o la AWS CLI.

### Important

Este procedimiento configura el bucket de S3 para invocar su función cada vez que se crea un objeto en el bucket. Asegúrese de configurar esto solo en el bucket de origen. Si la función de Lambda crea objetos en el mismo bucket que la invoca, la función se puede [invocar de forma continua en un bucle](#). Esto puede provocar que se facturen cargos imprevistos en su Cuenta de AWS.

## Console

Para configurar el desencadenador de Amazon S3 (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija la función (EncryptPDF).
2. Elija Add trigger (Añadir disparador).
3. Seleccione S3.
4. En Bucket, seleccione el bucket de origen.
5. En Tipos de eventos, seleccione Todos los eventos de creación de objetos.
6. En Recursive invocation (Invocación recursiva), marque la casilla de verificación para confirmar que no se recomienda utilizar el mismo bucket de S3 para la entrada y la salida. Puede obtener más información sobre los patrones de invocación recursiva en Lambda en [Patrones recursivos que provocan funciones de Lambda descontroladas](#) en Serverless Land.
7. Elija Añadir.

Al crear un desencadenador mediante la consola de Lambda, Lambda crea automáticamente una [política basada en recursos](#) para conceder permiso al servicio que seleccione para invocar la función.


## AWS CLI

Para configurar el desencadenador de Amazon S3 (AWS CLI)

1. Para que el bucket de origen de Amazon S3 invoque la función cuando agregue un archivo de imagen, primero debe configurar los permisos de la función mediante una [política basada en recursos](#). Una instrucción de política basada en recursos concede permisos a otros Servicios de AWS para invocar la función. Para conceder permisos a Amazon S3 para invocar la función, ejecute el siguiente comando de la CLI. Asegúrese de reemplazar el parámetro `source-account` por su propio ID de Cuenta de AWS y de utilizar su propio nombre de bucket de origen.

```
aws lambda add-permission --function-name EncryptPDF \  
--principal s3.amazonaws.com --statement-id s3invoke --action \  
"lambda:InvokeFunction" \  
--source-arn arn:aws:s3:::SOURCEBUCKET \  
--source-account 123456789012
```

La política que defina con este comando permite a Amazon S3 invocar la función solo cuando se lleva a cabo una acción en el bucket de origen.

 Note

Si bien los nombres de los buckets de S3 son únicos a nivel mundial, cuando se utilizan políticas basadas en recursos, se recomienda aclarar que el bucket debe pertenecer a la cuenta. Esto sucede porque si elimina un bucket, es posible que otra Cuenta de AWS cree un bucket con el mismo Nombre de recurso de Amazon (ARN).

2. Guarde el siguiente JSON en un archivo llamado `notification.json`. Cuando se aplica al bucket de origen, este JSON configura el bucket para enviar una notificación a la función de Lambda cada vez que se agrega un objeto nuevo. Reemplace el número de Cuenta de AWS y la Región de AWS en el ARN de la función de Lambda por su número de cuenta y su región.

```
{
  "LambdaFunctionConfigurations": [
    {
      "Id": "EncryptPDFEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:EncryptPDF",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Ejecute el siguiente comando de la CLI para aplicar la configuración de notificación del archivo JSON que creó al bucket de origen. Reemplace `SOURCEBUCKET` por el nombre del bucket de origen.

```
aws s3api put-bucket-notification-configuration --bucket SOURCEBUCKET \
--notification-configuration file://notification.json
```

Para obtener más información sobre el comando `put-bucket-notification-configuration` y la opción `notification-configuration`, consulte [put-bucket-notification-configuration](#) en la Referencia de comandos de la CLI AWS.

## Implementación de los recursos mediante AWS SAM

Para implementar la aplicación de ejemplo con la CLI de AWS SAM, lleve a cabo los siguientes pasos:

Asegúrese de tener [instalada la última versión de la CLI](#) y de tener [Docker](#) instalado en la máquina de compilación.

1. Edite el archivo `template.yaml` para especificar el nombre de los buckets de S3. Los buckets de S3 deben tener nombres únicos a nivel mundial que cumplan con las [reglas de nomenclatura para los buckets S3](#).

Reemplace el nombre del bucket `EXAMPLE-BUCKET` con un nombre de su elección compuesto de letras minúsculas, números, puntos (.) y guiones (-). Para el bucket de destino, reemplace `EXAMPLE-BUCKET-encrypted` por `<source-bucket-name>-encrypted` cuando `<source-bucket>` sea el nombre que escogió para el bucket de origen.

2. Ejecute el comando a continuación en el directorio en el que guardó los archivos `template.yaml`, `lambda_function.py` y `requirements.txt`.

```
sam build --use-container
```

Este comando recopila los artefactos de compilación de la aplicación y los coloca en el formato y la ubicación adecuados para implementarlos. Con la opción `--use-container`, la función se crea dentro de un contenedor de Docker tipo Lambda. Lo usamos aquí para que no necesite tener Python 3.12 instalado en su máquina local para que la compilación funcione.

Durante el proceso de compilación, AWS SAM busca el código de la función de Lambda en la ubicación que especificó con la propiedad `CodeUri` en la plantilla. En este caso, especificamos el directorio actual como la ubicación (`./`).

Si hay un archivo `requirements.txt`, AWS SAM lo usa para recopilar las dependencias especificadas. De forma predeterminada, AWS SAM crea un paquete de implementación `.zip` con el código y las dependencias de la función. También puede optar por implementar la función como una imagen de contenedor mediante la propiedad [PackageType](#).

3. Para implementar la aplicación y crear los recursos de Lambda y Amazon S3 especificados en la plantilla de AWS SAM, ejecute el siguiente comando:

```
sam deploy --guided
```

El uso de la marca `--guided` significa que AWS SAM le mostrará instrucciones para guiarlo a lo largo del proceso de implementación. Para esta implementación, acepte las opciones predeterminadas pulsando Intro.

Durante el proceso de implementación, AWS SAM crea los siguientes recursos en su Cuenta de AWS:

- Una [pila](#) de AWS CloudFormation llamada `sam-app`
- Una función de Lambda con el nombre `EncryptPDF`
- Dos buckets de S3 con los nombres que eligió cuando editó el archivo de plantilla `template.yaml` de AWS SAM
- Un rol de ejecución de IAM para la función con el formato de nombre `sam-app-EncryptPDFFunctionRole-2qGaapHFWOQ8`

Cuando AWS SAM termine de crear los recursos, debería ver el siguiente mensaje:

```
Successfully created/updated stack - sam-app in us-west-2
```

## Pruebas de la aplicación

Para probar la aplicación, debe cargar un archivo PDF en el bucket de origen y confirmar que Lambda crea una versión cifrada del archivo en el bucket de destino. En este ejemplo, puede probarlo de forma manual mediante la consola o la AWS CLI o usando el script de prueba proporcionado.

Para las aplicaciones de producción, puede utilizar métodos y técnicas de prueba tradicionales, como las pruebas unitarias, para confirmar el correcto funcionamiento del código de la función de Lambda. La mejor práctica también es implementar pruebas como las del script de prueba proporcionado, que llevan a cabo pruebas de integración con recursos reales basados en la nube. Las pruebas de integración en la nube confirman que la infraestructura se desplegó correctamente y que los eventos fluyen entre los distintos servicios según lo esperado. Para obtener más información, consulte [Prueba de funciones sin servidor](#).

## Prueba de la aplicación de forma manual

Para probar la función de forma manual, agregue un archivo PDF a su bucket de origen de Amazon S3. Cuando agrega el archivo al bucket de origen, la función de Lambda se debe invocar de forma automática y debe almacenar una versión cifrada del archivo en el bucket de destino.

### Console

#### Prueba de la aplicación cargando un archivo (consola)

1. Para cargar un archivo PDF en el bucket de S3, haga lo siguiente:
  - a. Abra la página [Buckets](#) de la consola de Amazon S3 y elija el bucket de origen.
  - b. Seleccione Cargar.
  - c. Elija Agregar archivos y utilice el selector de archivos para elegir el archivo PDF que desea cargar.
  - d. Elija Abrir y, a continuación, Cargar.
2. Compruebe que Lambda haya guardado una versión cifrada del archivo PDF en el bucket de destino, de la siguiente manera:
  - a. Vuelva a la página [Buckets](#) de la consola de Amazon S3 y elija el bucket de destino.
  - b. En el panel Objetos, ahora debería ver un archivo con el formato de nombre `filename_encrypted.pdf` (en el que `filename.pdf` es el nombre del archivo que cargó en el bucket de origen). Para descargar el PDF cifrado, seleccione el archivo y luego elija Descargar.
  - c. Confirme que puede abrir el archivo descargado con la contraseña con la que lo protegió la función de Lambda (`my-secret-password`).

### AWS CLI

#### Prueba de la aplicación cargando un archivo (AWS CLI)

1. Desde el directorio que contiene el archivo PDF que desea cargar, ejecute el siguiente comando de la CLI: Reemplace el parámetro `--bucket` con el nombre del bucket de origen. Para los parámetros `--key` y `--body`, utilice el nombre de archivo del archivo de prueba.

```
aws s3api put-object --bucket SOURCEBUCKET --key test.pdf --body ./test.pdf
```



2. Compruebe que la función creó una versión cifrada del archivo y la guardó en el bucket de S3 de destino. Ejecute el siguiente comando de la CLI, para ello reemplace `SOURCEBUCKET-encrypted` por el nombre del bucket de destino.

```
aws s3api list-objects-v2 --bucket SOURCEBUCKET-encrypted
```

Si la función se ejecuta correctamente, verá un resultado similar al siguiente.

El bucket de destino debe contener un archivo con el formato de nombre

`<your_test_file>_encrypted.pdf`, en el que `<your_test_file>` es el nombre del archivo cargado.

```
{
  "Contents": [
    {
      "Key": "test_encrypted.pdf",
      "LastModified": "2023-06-07T00:15:50+00:00",
      "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
      "Size": 2763,
      "StorageClass": "STANDARD"
    }
  ]
}
```

3. Para descargar el archivo que Lambda guardó en el bucket de destino, ejecute el siguiente comando de la CLI: Reemplace el parámetro `--bucket` con el nombre del bucket de destino. Para el parámetro `--key`, utilice el nombre de archivo `<your_test_file>_encrypted.pdf`, en el que `<your_test_file>` es el nombre del archivo de prueba que cargó.

```
aws s3api get-object --bucket SOURCEBUCKET-encrypted --key test_encrypted.pdf
my_encrypted_file.pdf
```

Este comando descarga el archivo en el directorio actual y lo guarda como `my_encrypted_file.pdf`.

4. Confirme que puede abrir el archivo descargado con la contraseña con la que lo protegió la función de Lambda (`my-secret-password`).

## Prueba de la aplicación con el script automatizado

Para probar la aplicación con el script de prueba proporcionado, primero asegúrese de que el módulo `pytest` esté instalado en su entorno local. Puede instalar `pytest` ejecutando el siguiente comando:

```
pip install pytest
```

También debe editar el código del archivo `test_pdf_encrypt.py` para reemplazar los nombres de los buckets de los marcadores de posición por los nombres de los buckets de origen y destino de Amazon S3. Realice los siguientes cambios en `test_pdf_encrypt.py`:

- En la función `test_source_bucket_available`, reemplace `EXAMPLE-BUCKET` por el nombre de su bucket de origen.
- En la función `test_encrypted_file_in_bucket`, reemplace `EXAMPLE-BUCKET-encrypted` por `<source-bucket>-encrypted`, cuando `<source-bucket>` sea el nombre del bucket de origen.
- En la función `cleanup`, reemplace `EXAMPLE-BUCKET` por el nombre del bucket de origen y reemplace `EXAMPLE-BUCKET-encrypted` por `#source-bucket>-encrypted`, cuando `<source-bucket>` sea el nombre del bucket de origen.

Para ejecutar las pruebas, haga lo siguiente:

- Guarde un archivo PDF con el nombre `test.pdf` en el directorio que contiene los archivos `test_pdf_encrypt.py` y `pytest.ini`.
- Abra un programa de terminal o intérprete de comandos y ejecute el siguiente comando desde el directorio que contiene los archivos de prueba:

```
pytest -s -v
```

Cuando concluya la prueba, la salida debe tener el siguiente aspecto:

```
===== test session starts
=====
platform linux -- Python 3.12.2, pytest-7.2.2, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
```

```
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/pdf_encrypt_app/.hypothesis/examples')
Test order randomisation NOT enabled. Enable with --random-order or --random-order-bucket=<bucket_type>
rootdir: /home/pdf_encrypt_app, configfile: pytest.ini
plugins: anyio-3.7.1, hypothesis-6.70.0, localserver-0.7.1, random-order-1.1.0
collected 4 items

test_pdf_encrypt.py::test_source_bucket_available PASSED
test_pdf_encrypt.py::test_lambda_invoked PASSED
test_pdf_encrypt.py::test_encrypted_file_in_bucket PASSED
test_pdf_encrypt.py::test_cleanup PASSED
Deleted test.pdf from EXAMPLE-BUCKET
Deleted test_encrypted.pdf from EXAMPLE-BUCKET-encrypted

===== 4 passed in 7.32s
=====
```

## Siguientes pasos

Ahora que creó esta aplicación de ejemplo, puede usar el código proporcionado como base para crear otros tipos de aplicaciones de procesamiento de archivos. Modifique el código del archivo `lambda_function.py` para implementar la lógica de procesamiento de archivos para su caso de uso.

Muchos casos de uso típicos del procesamiento de archivos implican el procesamiento de imágenes. Cuando se usa Python, las bibliotecas de procesamiento de imágenes más populares, como [pillow](#), suelen contener componentes de C o C++. Para garantizar que el paquete de implementación de la función sea compatible con el entorno de ejecución de Lambda, es importante utilizar la distribución binaria de origen correcta.

Cuando implemente los recursos con AWS SAM, debe tomar algunas medidas adicionales para incluir la distribución de origen correcta en el paquete de implementación. Como AWS SAM no instalará dependencias para una plataforma diferente a la de la máquina de compilación, especificar la distribución de origen (archivo `.whl`) correcta en su archivo `requirements.txt` no funcionará si la máquina de compilación usa un sistema operativo o una arquitectura diferente del entorno de ejecución de Lambda. En su lugar, lleve a cabo una de las siguientes acciones:

- Use la opción `--use-container` cuando ejecute `sam build`. Cuando especifica esta opción, AWS SAM descarga una imagen base de contenedor compatible con el entorno de ejecución de Lambda y compila el paquete de implementación de la función en un contenedor de Docker con esa imagen. Para obtener más información, consulte [Creación de una función de Lambda dentro de un contenedor proporcionado](#).
- Cree usted mismo el paquete de implementación `.zip` de la función con la distribución binaria de origen correcta y guarde el archivo `.zip` en el directorio que especifique como `CodeUri` en la plantilla AWS SAM. Para obtener más información sobre la compilación de paquetes de implementación `.zip` para Python mediante distribuciones binarias, consulte [the section called “Creación de un paquete de despliegue `.zip` con dependencias”](#) y [the section called “Creación de paquetes de despliegue `.zip` con bibliotecas nativas”](#).

# Comprenda los conceptos clave de Lambda

Lambda es un servicio de computación basado en eventos que ejecuta instancias de una función para procesar eventos. Puede invocar su función directamente mediante la API de Lambda o puede configurar un servicio o recurso de AWS para invocarla.

En las siguientes secciones se describen algunos conceptos clave de las funciones de Lambda, el modelo de programación basado en eventos y el entorno en el que se ejecuta la función.

## Conceptos

- [Función](#)
- [Desencadenador](#)
- [Evento](#)
- [Entorno de ejecución](#)
- [Paquete de implementación](#)
- [Tiempo de ejecución](#)
- [Capa](#)
- [Simultaneidad](#)
- [Qualifier](#)
- [Destino](#)

## Función

Una función es un recurso que puede invocar para ejecutar el código en Lambda. Una función tiene código para procesar los [eventos](#) que pasa a la función o que otros AWS servicios envían a la función.

## Desencadenador

Un desencadenador es un recurso o configuración que invoca una función de Lambda. Los desencadenadores incluyen los servicios de AWS que puede configurar para invocar una función y [mapeos de fuentes de eventos](#). Un mapeo de fuente de eventos es un recurso de Lambda que lee elementos de un flujo o una cola e invoca una función. Para obtener más información, consulte

[Comprender los métodos de invocación de la función de Lambda](#) y [Invocar Lambda con eventos de otros servicios de AWS](#).

## Evento

Un evento es un documento con formato JSON que contiene datos para que una función de Lambda los procese. El tiempo de ejecución convierte el evento en un objeto y lo pasa al código de la función. Cuando se invoca una función, se determina la estructura y el contenido del evento.

Example evento personalizado: datos del tiempo

```
{
  "TemperatureK": 281,
  "WindKmh": -3,
  "HumidityPct": 0.55,
  "PressureHPa": 1020
}
```

Cuando un servicio de AWS invoca una función, el servicio define la forma del evento.

Example evento de servicio: notificación de Amazon SNS

```
{
  "Records": [
    {
      "Sns": {
        "Timestamp": "2019-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEKai6RibDsvpi+tE/1+82j...65r==",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        ...
      }
    }
  ]
}
```

Para obtener más información acerca de los eventos de los servicios de AWS, consulte [Invocar Lambda con eventos de otros servicios de AWS](#).

## Entorno de ejecución

Un entorno de ejecución proporciona un entorno en tiempo de ejecución seguro y aislado para su función de Lambda. Un entorno de ejecución administra los procesos y recursos necesarios para

ejecutar la función. El entorno de ejecución proporciona compatibilidad del ciclo de vida para la función y para cualquier [extensión](#) asociada a la función.

Para obtener más información, consulte [Comprender el ciclo de vida del entorno de ejecución de Lambda](#).

## Paquete de implementación

Implemente el código de las funciones de Lambda con un paquete de implementación. Lambda admite dos tipos de paquetes de implementaciones:

- Un archivo .zip que contiene su código de función y sus dependencias. Lambda proporciona el sistema operativo y el tiempo de ejecución de su función. Para obtener más información, consulte [Implementación de funciones de Lambda como archivos .zip](#).
- Imagen de contenedor compatible con la especificación [Open Container Initiative \(OCI\)](#). Agregue su código de función y dependencias a la imagen. También debe incluir el sistema operativo y un tiempo de ejecución de Lambda. Para obtener más información, consulte [Crear una función de Lambda con una imagen de contenedor](#).

## Tiempo de ejecución

El tiempo de ejecución proporciona un entorno específico del lenguaje que se ejecuta en el entorno de ejecución. El tiempo de ejecución transmite eventos de invocación, información de contexto y respuestas entre Lambda y la función. Puede usar tiempos de ejecución que Lambda proporcione, o crear los suyos propios. Si empaqueta el código como un archivo de archivo .zip, debe configurar su función para utilizar un tiempo de ejecución que coincida con su lenguaje de programación. Para una imagen contenedor, se incluye el tiempo de ejecución al compilar la imagen.

Para obtener más información, consulte [Tiempos de ejecución de Lambda](#).

## Capa

Una capa de Lambda es un archivo .zip que puede contener código u otros datos adicionales. Una capa puede contener bibliotecas, un [tiempo de ejecución personalizado](#), datos o archivos de configuración.

Las capas proporcionan una forma conveniente de empaquetar bibliotecas y otras dependencias que puede usar con las funciones de Lambda. El uso de capas reduce el tamaño de los archivos de

implementación cargados y acelera la implementación de su código. Las capas también promueven el uso compartido de código y la separación de responsabilidades para que pueda iterar más rápido al escribir la lógica empresarial.

Puede incluir hasta cinco capas por función. Las capas se contabilizan para las [cuotas del tamaño de implementación](#) estándar de Lambda. Cuando incluye una capa en una función, el contenido se extrae al directorio de `/opt` en el entorno de ejecución.

De forma predeterminada, las capas que cree son privadas de su cuenta de AWS. Puede optar por compartir una capa con otras cuentas o hacer que la capa sea pública. Si sus funciones consumen una capa que publicó una cuenta diferente, sus funciones pueden continuar usando la versión de la capa después de que se haya eliminado o después de que se revoque su permiso para acceder a la capa. Sin embargo, no puede crear una nueva función o actualizar funciones usando una versión de capa eliminada.

Las funciones implementadas como una imagen de contenedor no usan capas. En su lugar, empaqueta su tiempo de ejecución preferido, bibliotecas y otras dependencias en la imagen contenedor cuando construye la imagen.

Para obtener más información, consulte [Capas de Lambda](#).

## Simultaneidad

La simultaneidad es el número de solicitudes que la función atiende en un momento dado. Cuando se invoca la función, Lambda aprovisiona una instancia para procesar el evento. Cuando el código de la función termina de ejecutarse, puede encargarse de otra solicitud. Si la función se invoca de nuevo mientras se sigue procesando una solicitud, se aprovisiona otra instancia, lo que aumenta la simultaneidad de la función.

La simultaneidad está sujeta a [cuotas](#) en el nivel de región de AWS. Puede configurar funciones individuales para limitar su simultaneidad o para permitirles alcanzar un nivel específico de simultaneidad. Para obtener más información, consulte [Configurar la simultaneidad reservada para una función](#).

## Qualifier

Al invocar o consultar una función, puede incluir un calificador para especificar una versión o alias. Una versión es una instantánea inmutable del código y la configuración de una función que tiene un



calificador numérico. Por ejemplo, `my-function:1`. Un alias es un puntero a una versión que se puede actualizar para asignarla a una versión diferente o dividir el tráfico entre dos versiones. Por ejemplo, `my-function:BLUE`. Puede usar versiones y alias al mismo tiempo para proporcionar una interfaz estable para que los clientes invoquen su función.

Para obtener más información, consulte [Administrar las versiones de la función de Lambda](#).

## Destino

Un destino es un recurso de AWS en que Lambda puede enviar eventos desde una invocación asíncrona. Puede configurar un destino para eventos que no han podido procesarse. Algunos servicios también admiten un destino para eventos que se procesan correctamente.

Para obtener más información, consulte [Configuración de destinos para invocación asíncrona](#).

# Uso de Lambda con la infraestructura como código (IaC)

Lambda ofrece varias formas de implementar código y crear funciones. Por ejemplo, puede usar la consola de Lambda o la AWS Command Line Interface (AWS CLI) para crear o actualizar las funciones de Lambda manualmente. Además de estas opciones manuales, AWS ofrece una serie de soluciones para implementar funciones de Lambda y aplicaciones sin servidor con infraestructura como código (IaC). Con la IaC, puede aprovisionar y mantener las funciones de Lambda y otros recursos de AWS con código, en lugar de utilizar procesos y ajustes manuales.

La mayoría de las veces, las funciones de Lambda no se ejecutan de forma aislada. En cambio, forman parte de una aplicación sin servidor con otros recursos, como bases de datos, colas y almacenamiento. Con la IaC, puede automatizar sus procesos de implementación para implementar y actualizar de forma rápida y repetible aplicaciones sin servidor completas que requieren muchos recursos de AWS independientes. Este enfoque acelera el ciclo de desarrollo, facilita la administración de la configuración y garantiza que sus recursos se implementen siempre de la misma manera.

## Temas

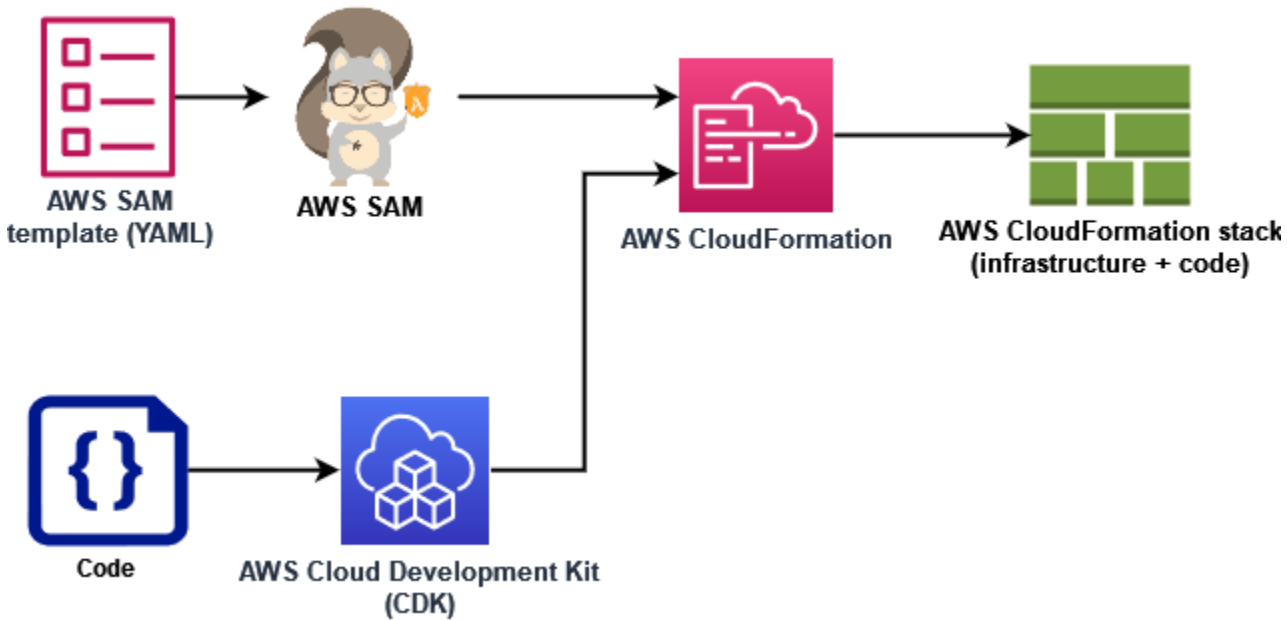
- [Herramientas de IaC para Lambda](#)
- [Introducción a IaC para Lambda](#)
- [Siguiendo los pasos](#)
- [Regiones en las que se admite la integración de Lambda a Application Composer](#)

## Herramientas de IaC para Lambda

Para implementar funciones de Lambda y aplicaciones sin servidor con IaC, AWS ofrece una serie de herramientas y servicios diferentes.

AWS CloudFormation fue el primer servicio que AWS ofreció para crear y configurar recursos en la nube. Con AWS CloudFormation, puede crear plantillas de texto para definir la infraestructura y el código. A medida que AWS introducía más servicios nuevos y se complicaba cada vez más la creación de plantillas de AWS CloudFormation, se lanzaron dos herramientas más. AWS SAM es otro marco basado en plantillas que se usa para definir aplicaciones sin servidor. El AWS Cloud Development Kit (AWS CDK) representa un enfoque basado en el código para definir y aprovisionar la infraestructura mediante constructos de código en muchos lenguajes de programación populares.

Con tanto AWS SAM como el AWS CDK, AWS CloudFormation se encarga de las tareas no visibles para crear e implementar su infraestructura. En el siguiente diagrama, se muestra la relación entre estas herramientas y, en los párrafos posteriores, se explican sus características clave.



- **AWS CloudFormation:** con CloudFormation, modela y configura sus recursos de AWS a partir de una plantilla YAML o JSON que describe los recursos y sus propiedades. CloudFormation aprovisiona sus recursos de forma segura y repetible, lo que le permite crear su infraestructura y aplicaciones sin pasos manuales y con frecuencia. Si cambia la configuración, CloudFormation determina cuáles son las operaciones correctas que se deben realizar para actualizar la pila. CloudFormation puede incluso revertir los cambios.
- **AWS Serverless Application Model (AWS SAM):** AWS SAM es un marco de código abierto que sirve para definir aplicaciones sin servidor. Las plantillas de AWS SAM utilizan una sintaxis abreviada para definir funciones, API, bases de datos y asignaciones de orígenes de eventos con solo unas pocas líneas de texto (YAML) por recurso. Durante la implementación, AWS SAM transforma y amplía la sintaxis de AWS SAM para convertirla en sintaxis de AWS CloudFormation. Por ello, se puede agregar cualquier sintaxis de CloudFormation a las plantillas de AWS SAM. Esto proporciona a AWS SAM todas las capacidades de CloudFormation, pero con menos líneas de código para la configuración.
- **AWS Cloud Development Kit (AWS CDK):** con el AWS CDK, define su infraestructura mediante constructos de código y la aprovisiona con AWS CloudFormation. El AWS CDK le permite modelar la infraestructura de aplicaciones con TypeScript, Python, Java, .NET y Go (en la versión preliminar para desarrolladores) con el entorno de desarrollo integrado, las herramientas de

prueba y los patrones de flujo de trabajo que tenga. Obtendrá todos los beneficios que ofrece AWS CloudFormation, entre los que se incluyen implementación repetible, reversión sencilla de cambios y detección de desviaciones.

AWS también ofrece un servicio llamado AWS Application Composer para desarrollar plantillas de IaC mediante una interfaz gráfica sencilla. Con Application Composer, puede diseñar una arquitectura de aplicaciones arrastrando, agrupando y conectando los Servicios de AWS en un lienzo visual. A continuación, Application Composer crea una plantilla de AWS SAM o una plantilla de AWS CloudFormation a partir de su diseño que puede usar para implementar la aplicación.

En la siguiente sección, [the section called “Introducción a IaC para Lambda”](#), utilizará Application Composer para desarrollar una plantilla para una aplicación sin servidor basada en una función de Lambda existente.

## Introducción a IaC para Lambda

En este tutorial, puede empezar a utilizar IaC con Lambda al crear una plantilla de AWS SAM a partir de una función de Lambda existente y, a continuación, crear una aplicación sin servidor en Application Composer al agregar otros recursos de AWS.

Si prefiere empezar con un tutorial de AWS SAM o AWS CloudFormation para aprender a trabajar con plantillas sin usar Application Composer, encontrará enlaces a otros recursos en la sección [the section called “Siguiendo pasos”](#) al final de esta página.

A medida que realice este tutorial, aprenderá algunos conceptos fundamentales, como la forma en que se especifican los recursos de AWS en AWS SAM. También aprenderá a usar Application Composer para crear una aplicación sin servidor que pueda implementar usando AWS SAM o AWS CloudFormation.

Para completar este tutorial, debe llevar a cabo los siguientes pasos:

- Crear un ejemplo de función de Lambda
- Utilizar la consola de Lambda para obtener la plantilla de AWS SAM de la función
- Exportar la configuración de la función a AWS Application Composer y diseñar una aplicación sin servidor sencilla que se base en la configuración de la función
- Guardar una plantilla de AWS SAM actualizada que pueda utilizar como base para implementar su aplicación sin servidor

En la sección [the section called “Siguientes pasos”](#), encontrará recursos que puede utilizar para obtener más información sobre AWS SAM y Application Composer. Estos recursos incluyen enlaces a tutoriales más avanzados que le enseñan a implementar una aplicación sin servidor con AWS SAM.

## Requisitos previos

En este tutorial, utilizará la característica de [sincronización local](#) de Application Composer para guardar la plantilla y los archivos de código en su máquina de compilación local. Para utilizar esta característica, necesita un navegador compatible con la API de acceso al sistema de archivos, que permite a las aplicaciones web leer, escribir y guardar archivos en su sistema de archivos local. Recomendamos utilizar Google Chrome o Microsoft Edge. Para obtener más información sobre la API de acceso al sistema de archivos, consulte [¿Qué es la API de acceso al sistema de archivos?](#)

## Creación de una función de Lambda

En este primer paso, creará una función de Lambda que podrá utilizar para completar el resto del tutorial. Para simplificar las tareas, utiliza la consola de Lambda para crear una función básica “Hola, mundo” con el tiempo de ejecución Python 3.11.

Para crear una función de Lambda “Hola, mundo” con la consola

1. Abra la [consola de Lambda](#).
2. Elija Crear función.
3. Deje seleccionada la opción Crear desde cero y, en Información básica, ingrese **LambdaIaCDemo** en Nombre de la función.
4. En Tiempo de ejecución, elija Python 3.11.
5. Elija Crear función.

## Visualización de la plantilla de AWS SAM para su función

Antes de exportar la configuración de la función a Application Composer, utilice la consola de Lambda para ver la configuración actual de la función como una plantilla de AWS SAM. Si sigue los pasos de esta sección, aprenderá sobre la anatomía de una plantilla de AWS SAM y cómo definir recursos, como las funciones de Lambda, para empezar a especificar una aplicación sin servidor.

## Visualización de la plantilla de AWS SAM para su función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función que acaba de crear (LambdaIaCDemo).
3. En el panel Información general de la función, elija Plantilla.

En lugar del diagrama que representa la configuración de la función, verá una plantilla de AWS SAM para la función. La plantilla debe tener el siguiente aspecto.

```
# This AWS SAM template has been generated from your function's
# configuration. If your function has one or more triggers, note
# that the AWS resources associated with these triggers aren't fully
# specified in this template and include placeholder values. Open this template
# in AWS Application Composer or your favorite IDE and modify
# it to specify a serverless application with other AWS resources.
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        Statement:
```

```
- Effect: Allow
  Action:
    - logs:CreateLogGroup
  Resource: arn:aws:logs:us-east-1:123456789012:*
- Effect: Allow
  Action:
    - logs:CreateLogStream
    - logs:PutLogEvents
  Resource:
    - >-
      arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
LambdaIaCDemo:*
```

Dediquemos un momento a analizar la plantilla YAML de su función y comprender algunos conceptos clave.

La plantilla comienza con la declaración `Transform: AWS::Serverless-2016-10-31`. Esta declaración es obligatoria porque, en segundo plano, las plantillas de AWS SAM se implementan con AWS CloudFormation. Usar la instrucción `Transform` identifica la plantilla como un archivo de plantilla de AWS SAM.

Después de la declaración `Transform`, viene la sección `Resources`. Aquí es donde se definen los recursos de AWS que desea implementar con su plantilla de AWS SAM. Las plantillas de AWS SAM pueden contener una combinación de recursos de AWS SAM y AWS CloudFormation. Esto se debe a que, durante la implementación, las plantillas de AWS SAM se expanden para volverlas plantillas de AWS CloudFormation, por lo que se puede agregar cualquier sintaxis de AWS CloudFormation válida a una plantilla de AWS SAM.

Por el momento, solo hay un recurso definido en la sección `Resources` de la plantilla, la función de Lambda `LambdaIaCDemo`. Para agregar una función de Lambda a una plantilla de AWS SAM, utilice el tipo de recurso `AWS::Serverless::Function`. Las `Properties` de una función de Lambda definen el tiempo de ejecución de la función, su controlador y otras opciones de configuración. La ruta al código fuente de la función que AWS SAM debe usar para implementarla también se define aquí. Para obtener más información sobre los recursos de funciones de Lambda en AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS SAM.

Además de las propiedades y las configuraciones de la función, la plantilla también especifica una política de AWS Identity and Access Management (IAM) para su función. Esta política otorga permiso a su función para escribir registros en Registros de Amazon CloudWatch. Al crear una función en la

consola de Lambda, dicho servicio adjunta esta política a la función automáticamente. Para obtener más información sobre cómo especificar una política de IAM para una función en una plantilla de AWS SAM, consulte la propiedad `policies` en la página [AWS::Serverless::Function](#) de la Guía para desarrolladores de AWS SAM.

Para obtener más información sobre la estructura de plantillas de AWS SAM, consulte [AWS SAM template anatomy](#).

## Uso de AWS Application Composer para diseñar una aplicación sin servidor

Para empezar a crear una aplicación sin servidor sencilla con la plantilla de AWS SAM de la función como punto de partida, exporte la configuración de la función a Application Composer y active el modo de sincronización local de Application Composer. La sincronización local guarda automáticamente el código de la función y la plantilla de AWS SAM en la máquina de compilación local y mantiene la plantilla guardada sincronizada a medida que agrega otros recursos de AWS a Application Composer.

### Exportación de la función a Application Composer

1. En el panel Información general de la función, elija Exportar a Application Composer.

Para exportar la configuración y el código de la función a Application Composer, Lambda crea un bucket de Amazon S3 en su cuenta para almacenar estos datos temporalmente.

2. En el cuadro de diálogo, seleccione Confirmar y crear proyecto para aceptar el nombre predeterminado de este bucket y exportar la configuración y el código de la función a Application Composer.
3. (Opcional) Para elegir otro nombre para el bucket de Amazon S3 que Lambda crea, introduzca un nombre nuevo y elija Confirmar y crear proyecto. Los nombres de los buckets de Amazon S3 no pueden repetirse en ningún lado y deben seguir las [reglas de nomenclatura de buckets](#).

Al seleccionar Confirmar y crear proyecto, se abre la consola de Application Composer. En el lienzo, verá su función de Lambda.

4. En el menú desplegable, seleccione Activar la sincronización local.
5. En el cuadro de diálogo que se abre, elija Seleccionar carpeta y, luego, seleccione una carpeta de su máquina de compilación local.
6. Seleccione Activar para activar la sincronización local.



A fin de exportar su función a Application Composer, necesita permiso para usar ciertas acciones de la API. Si no puede exportar la función, compruebe [the section called “Permisos necesarios”](#) y asegúrese de tener los permisos que necesita.

### Note

Para el bucket que Lambda crea, se aplican los [precios de Amazon S3](#) estándar al exportar una función a Application Composer. Los objetos que Lambda coloca en el bucket se eliminan automáticamente después de 10 días, pero Lambda no elimina el bucket en sí. Para evitar que se agreguen más cargos a su Cuenta de AWS, siga las instrucciones que se indican en [Eliminar un bucket](#) después de haber exportado la función a Application Composer. Para obtener más información acerca del bucket de Amazon S3 que Lambda crea, consulte [the section called “Application Composer”](#).

## Diseño de su aplicación sin servidor en Application Composer

Después de activar la sincronización local, los cambios que realice en Application Composer se reflejarán en la plantilla de AWS SAM guardada en su máquina de compilación local. Ahora puede arrastrar y soltar recursos de AWS adicionales en el lienzo de Application Composer para crear su aplicación. En este ejemplo, agrega una cola simple de Amazon SQS como desencadenador de la función de Lambda y una tabla de DynamoDB en la que la función escribe datos.

1. Para agregar un desencadenador de Amazon SQS a su función de Lambda, haga lo siguiente:
  - a. En el campo de búsqueda de la paleta Recursos, ingrese **SQS**.
  - b. Arrastre el recurso Cola de SQS al lienzo y colóquelo a la izquierda de la función de Lambda.
  - c. Elija Detalles y, en ID lógico, ingrese **LambdaIaCQueue**.
  - d. Seleccione Guardar.
  - e. Conecte sus recursos de Amazon SQS y Lambda haciendo clic en el puerto Suscripción de la tarjeta de cola de SQS y arrastrándolo hasta el puerto izquierdo de la tarjeta de función de Lambda. La aparición de una línea entre los dos recursos indica que la conexión se ha realizado correctamente. Application Composer también muestra un mensaje en la parte inferior del lienzo que indica que los dos recursos se han conectado correctamente.
2. Para agregar una tabla de Amazon DynamoDB en la que la función de Lambda pueda escribir datos, haga lo siguiente:

- a. En el campo de búsqueda de la paleta Recursos, ingrese **DynamoDB**.
- b. Arrastre el recurso Tabla de DynamoDB al lienzo y colóquelo a la derecha de la función de Lambda.
- c. Elija Detalles y, en ID lógico, ingrese **LambdaIaCTable**.
- d. Seleccione Guardar.
- e. Conecte la tabla de DynamoDB a la función de Lambda haciendo clic en el puerto a la derecha de la tarjeta de la función de Lambda y arrastrándolo hasta el puerto a la izquierda de la tarjeta de DynamoDB.

Ahora que ya ha agregado estos recursos adicionales, echemos un vistazo a la plantilla actualizada de AWS SAM que ha creado Application Composer.

Visualización de la plantilla de AWS SAM actualizada

- En el lienzo de Application Composer, elija Plantilla para cambiar de la vista de lienzo a la vista de plantilla.

Su plantilla de AWS SAM ya debería contener los siguientes recursos y propiedades adicionales:

- Una cola de Amazon SQS con el identificador LambdaIaCQueue

```
LambdaIaCQueue:  
  Type: AWS::SQS::Queue  
  Properties:  
    MessageRetentionPeriod: 345600
```

Al agregar una cola de Amazon SQS mediante Application Composer, este establece la propiedad `MessageRetentionPeriod`. También puede establecer la propiedad `FifoQueue` seleccionando Detalles en la tarjeta de cola de SQS y marcando o desmarcando la cola FIFO.

Para establecer otras propiedades de la cola, puede editar la plantilla manualmente para agregarlas. Para obtener más información sobre el recurso `AWS::SQS::Queue` y sus propiedades disponibles, consulte [AWS::SQS::Queue](#) en la Guía del usuario de AWS CloudFormation.

- Una propiedad `Events` en la definición de la función de Lambda que especifica la cola de Amazon SQS como desencadenador de la función

```

Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1

```

La propiedad `Events` consta de un tipo de evento y un conjunto de propiedades que dependen del tipo. Para obtener información sobre los distintos Servicios de AWS que puede configurar para activar una función de Lambda y las propiedades que puede establecer, consulte [EventSource](#) en la Guía para desarrolladores de AWS SAM.

- Una tabla de DynamoDB con el identificador `LambdaIaCTable`

```

LambdaIaCTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: id
        AttributeType: S
    BillingMode: PAY_PER_REQUEST
    KeySchema:
      - AttributeName: id
        KeyType: HASH
    StreamSpecification:
      StreamViewType: NEW_AND_OLD_IMAGES

```

Al agregar una tabla de DynamoDB mediante Application Composer, puede configurar las claves de la tabla seleccionando Detalles en la tarjeta de la tabla de DynamoDB y editando los valores de la clave. Application Composer también establece los valores predeterminados para otras propiedades, incluidas `BillingMode` y `StreamViewType`.

Para obtener más información sobre estas y otras propiedades que puede agregar a la plantilla de AWS SAM, consulte [AWS::DynamoDB::Table](#) en la Guía del usuario de AWS CloudFormation.

- Una nueva política de IAM que permite a su función realizar operaciones CRUD en la tabla de DynamoDB que ha agregado

```

Policies:
  ...
  - DynamoDBCrudPolicy:

```

```
TableName: !Ref LambdaIaCTable
```

La plantilla final de AWS SAM debería verse similar al siguiente ejemplo.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        - Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
              Resource: arn:aws:logs:us-east-1:594035263019:*
            - Effect: Allow
              Action:
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource:
                - arn:aws:logs:us-east-1:594035263019:log-group:/aws/lambda/
LambdaIaCDemo:*
```

```

- DynamoDBCrudPolicy:
  TableName: !Ref LambdaIaCTable
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1
  Environment:
    Variables:
      LAMBDAIACTABLE_TABLE_NAME: !Ref LambdaIaCTable
      LAMBDAIACTABLE_TABLE_ARN: !GetAtt LambdaIaCTable.Arn
LambdaIaCQueue:
  Type: AWS::SQS::Queue
  Properties:
    MessageRetentionPeriod: 345600
LambdaIaCTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: id
        AttributeType: S
    BillingMode: PAY_PER_REQUEST
    KeySchema:
      - AttributeName: id
        KeyType: HASH
    StreamSpecification:
      StreamViewType: NEW_AND_OLD_IMAGES

```

## Implementación de su aplicación sin servidor mediante AWS SAM (opcional)

Si desea utilizar AWS SAM para implementar una aplicación sin servidor con la plantilla que acaba de crear en Application Composer, primero debe instalar la AWS SAM CLI. Para ello, siga las instrucciones que se enumeran en [Instalación de la AWS SAM CLI](#).

Antes de implementar la aplicación, también debe actualizar el código de la función que Application Composer guardó junto con la plantilla. Por el momento, el archivo `lambda_function.py` que guardó Application Composer contiene solo el código básico “Hola, mundo” que Lambda proporcionó al crear la función.

Para actualizar el código de la función, copie el siguiente código y péguelo en el archivo `lambda_function.py` que Application Composer guardó en su máquina de compilación local. Cuando activó el modo de sincronización local, especificó el directorio en el que Application Composer debe guardar este archivo.

Este código acepta un par clave-valor en un mensaje de la cola de Amazon SQS que creó en Application Composer. Si tanto la clave como el valor son cadenas, el código las utiliza para escribir un elemento en la tabla de DynamoDB definida en la plantilla.

### Código de la función Python actualizado

```
import boto3
import os
import json

# define the DynamoDB table that Lambda will connect to
tablename = os.environ['LAMBDAIACTABLE_TABLE_NAME']

# create the DynamoDB resource
dynamo = boto3.client('dynamodb')

def lambda_handler(event, context):
    # get the message out of the SQS event
    message = event['Records'][0]['body']
    data = json.loads(message)
    # write event data to DDB table
    if check_message_format(data):
        key = next(iter(data))
        value = data[key]
        dynamo.put_item(
            TableName=tablename,
            Item={
                'id': {'S': key},
                'Value': {'S': value}
            }
        )
    else:
        raise ValueError("Input data not in the correct format")

# check that the event object contains a single key value
# pair that can be written to the database
def check_message_format(message):
    if len(message) != 1:
```

```
        return False

    key, value = next(iter(message.items()))

    if not (isinstance(key, str) and isinstance(value, str)):
        return False

    else:
        return True
```

## Implementación de la aplicación sin servidor

Para implementar su aplicación con la AWS SAM CLI, lleve a cabo los siguientes pasos. Para que su función se compile e implemente correctamente, su máquina de compilación y su PATH deben tener instalada la versión 3.11 de Python.

1. Ejecute el siguiente comando desde el directorio en el que Application Composer guardó sus archivos `template.yaml` y `lambda_function.py`.

```
sam build
```

Este comando recopila los artefactos de compilación de la aplicación y los coloca en el formato y la ubicación adecuados para implementarlos.

2. Para implementar la aplicación y crear los recursos de Lambda, Amazon SQS y DynamoDB especificados en la plantilla de AWS SAM, ejecute el siguiente comando.

```
sam deploy --guided
```

El uso de la marca `--guided` significa que AWS SAM le mostrará instrucciones para guiarlo a lo largo del proceso de implementación. Para esta implementación, acepte las opciones predeterminadas pulsando Intro.

Durante el proceso de implementación, AWS SAM crea los siguientes recursos en su Cuenta de AWS:

- Una [pila](#) de AWS CloudFormation llamada `sam-app`
- Una función de Lambda con el formato de nombre `sam-app-LambdaIaCDemo-99VXPpYQVv1M`

- Una cola de Amazon SQS con el formato de nombre `sam-app-LambdaIaCQueue-xL87VeKsGiIo`
- Una tabla de DynamoDB con el formato de nombre `sam-app-LambdaIaCTable-CN0S66C0VLNV`

AWS SAM también crea las políticas y los roles de IAM necesarios para que su función de Lambda pueda leer los mensajes de la cola de Amazon SQS y realizar operaciones CRUD en la tabla de DynamoDB.

Para obtener más información sobre cómo implementar aplicaciones sin servidor con AWS SAM, consulte los recursos de la sección [the section called “Siguiendo pasos”](#).

## Puesta a prueba de la aplicación implementada (opcional)

Para confirmar que la aplicación sin servidor se ha implementado correctamente, envíe un mensaje a la cola de Amazon SQS que contenga un par clave-valor y compruebe que Lambda escribe un elemento en la tabla de DynamoDB con estos valores.

Para probar la aplicación sin servidor

1. Abra la página [Colas](#) de la consola de Amazon SQS y seleccione la cola que creó AWS SAM en su plantilla. El nombre tiene el formato `sam-app-LambdaIaCQueue-xL87VeKsGiIo`.
2. Elija Enviar y recibir mensajes y pegue el siguiente JSON en el Cuerpo del mensaje de la sección Enviar mensaje.

```
{
  "myKey": "myValue"
}
```

3. Elija Enviar mensaje.

Al enviar el mensaje a la cola, Lambda invocará la función a través de la asignación de orígenes de eventos definida en la plantilla de AWS SAM. Para confirmar que Lambda haya invocado su función según lo previsto, confirme que se haya agregado un elemento a la tabla de DynamoDB.

4. Abra la página [Tablas](#) en la consola de DynamoDB y seleccione su tabla. El nombre tiene el formato `sam-app-LambdaIaCTable-CN0S66C0VLNV`.
5. Elija Explorar elementos de la tabla. En el panel Devolución de elementos, debería ver un elemento con el id `myKey` y el valor `myValue`.



## Siguientes pasos

Para obtener más información sobre el uso de Application Composer con AWS SAM y AWS CloudFormation, comience con el artículo [Uso de Application Composer con AWS CloudFormation y AWS SAM](#).

Para ver un tutorial guiado en el que se utiliza AWS SAM para implementar una aplicación sin servidor diseñada en Application Composer, también le recomendamos que realice el [Tutorial de AWS Application Composer](#) en el [Taller AWS Serverless Patterns](#).

AWS SAM proporciona una interfaz de línea de comandos (CLI) que puede utilizar con plantillas de AWS SAM e integraciones de terceros compatibles para crear y ejecutar sus aplicaciones sin servidor. Con la AWS SAM CLI, puede crear e implementar su aplicación, realizar pruebas y depuraciones locales, configurar canalizaciones de CI/CD y mucho más. Para obtener más información sobre el uso de AWS SAM CLI, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

Para aprender a implementar una aplicación sin servidor con una plantilla de AWS SAM mediante la consola de AWS CloudFormation, comience con la sección [Uso de la consola de AWS CloudFormation](#) en la Guía del usuario de AWS CloudFormation.

## Regiones en las que se admite la integración de Lambda a Application Composer

La integración de Lambda a Application Composer se admite en las siguientes Regiones de AWS:

- Este de EE. UU. (Norte de Virginia)
- Este de EE. UU. (Ohio)
- Oeste de EE. UU. (Norte de California)
- Oeste de EE. UU. (Oregón)
- África (Ciudad del Cabo)
- Asia-Pacífico (Hong Kong)
- Asia-Pacífico (Hyderabad)
- Asia-Pacífico (Yakarta)
- Asia-Pacífico (Melbourne)
- Asia-Pacífico (Bombay)

- Asia-Pacífico (Osaka)
- Asia-Pacífico (Seúl)
- Asia-Pacífico (Singapur)
- Asia-Pacífico (Sídney)
- Asia-Pacífico (Tokio)
- Canadá (centro)
- Europa (Fráncfort)
- Europa (Zúrich)
- Europa (Irlanda)
- Europe (London)
- Europa (Estocolmo)
- Oriente Medio (EAU)

# Cómo entender el modelo de programación de Lambda

Lambda proporciona un modelo de programación que es común a todos los tiempos de ejecución. El modelo de programación define la interfaz entre el código y el sistema de Lambda. Le indica a Lambda el punto de entrada a la función al definir un controlador en la configuración de la función. El runtime se pasa en forma de objetos al controlador que contiene el evento de invocación y el contexto, como el nombre de la función y el ID de la solicitud.

Cuando el controlador termina de procesar el primer evento, el runtime le envía otro. La clase de la función permanece en la memoria, por lo que se pueden reutilizar los clientes y las variables declarados fuera del método del controlador en código de inicialización. Para ahorrar tiempo de procesamiento en eventos posteriores, cree recursos reutilizables como clientes AWSSDK durante la inicialización. Una vez inicializada, cada instancia de su función puede procesar miles de solicitudes.

La función también tiene acceso al almacenamiento local en el directorio `/tmp`. El contenido del directorio se conserva al congelar el entorno de ejecución, proporcionando una caché transitoria que se puede utilizar para varias invocaciones. Para obtener más información, consulte [Entorno de ejecución de Lambda](#).

Cuando se habilita el [seguimiento de AWS X-Ray](#), el tiempo de ejecución registra subsegmentos separados para la inicialización y la ejecución.

El tiempo de ejecución captura la salida de registro de la función y la envía a Amazon CloudWatch Logs. Además de registrar la salida de la función, el tiempo de ejecución también registra las entradas cuando comienza y finaliza la invocación. Esto incluye un registro de informe con el ID de solicitud, la duración facturada, la duración de inicialización y otros detalles. Si la función genera un error, el runtime devuelve ese error al invocador.

## Note

El registro está sujeto a [Cuotas de CloudWatch Logs](#). Los datos de registro se pueden perder debido a una limitación controlada o, en algunos casos, cuando se detiene una instancia de su función.

Lambda escala la función ejecutando instancias adicionales a medida que aumenta la demanda y deteniendo instancias a medida que disminuye la demanda. Este modelo conduce a variaciones en la arquitectura de la aplicación, por ejemplo:

- A menos que se indique lo contrario, las solicitudes entrantes pueden procesarse de forma desordenada o simultánea.
- Almacene el estado de la aplicación en otra parte; no confíe en que las instancias de la función tendrán una larga vida útil.
- Utilice el almacenamiento local y los objetos de nivel de clase para aumentar el rendimiento, pero mantenga al mínimo el tamaño del paquete de implementación y la cantidad de datos que transfiere al entorno de ejecución.

Para obtener una introducción práctica sobre estos conceptos en su lenguaje de programación preferido, consulte los siguientes capítulos.

- [Creación de funciones de Lambda con Node.js](#)
- [Creación de funciones de Lambda con Python](#)
- [Creación de funciones de Lambda con Ruby](#)
- [Creación de funciones de Lambda con Java](#)
- [Creación de funciones de Lambda con Go](#)
- [Creación de funciones Lambda con C#](#)
- [Creación de funciones de Lambda con PowerShell](#)

# Tiempos de ejecución de Lambda

Lambda admite múltiples idiomas a través del uso de tiempos de ejecución. Un tiempo de ejecución proporciona un entorno específico del lenguaje que transmite eventos de invocación, información de contexto y respuestas entre Lambda y la función. Puede usar tiempos de ejecución que Lambda proporcione, o crear los suyos propios.

Cada versión principal del lenguaje de programación tiene un tiempo de ejecución independiente, con un identificador de tiempo de ejecución único, como `nodejs20.x` o `python3.12`. Para configurar una función a fin de que utilice una nueva versión principal del lenguaje, debe cambiar el identificador del tiempo de ejecución. Como AWS Lambda no puede garantizar la compatibilidad con versiones anteriores entre las versiones principales, esta es una operación que depende del cliente.

Para una [función definida como una imagen de contenedor](#), se elige un tiempo de ejecución y la distribución de Linux al crear la imagen contenedor. Para cambiar el tiempo de ejecución, cree una nueva imagen contenedor.

Cuando se utiliza un archivo de archivo `.zip` para el paquete de implementación, se elige un tiempo de ejecución al crear la función. Para cambiar el tiempo de ejecución, puede [actualizar la configuración de su función](#). El tiempo de ejecución está emparejado con una de las distribuciones de Amazon Linux. El entorno de ejecución subyacente ofrece [variables de entorno](#) y bibliotecas adicionales a las que puede acceder desde el código de función.

Lambda invoca la función en un [entorno de ejecución](#). El entorno de ejecución proporciona un entorno en tiempo de ejecución seguro y aislado que administra los recursos necesarios para ejecutar la función. Lambda reutiliza el entorno de ejecución de una invocación previa, si la hay, o puede crear un nuevo entorno de ejecución.

Para usar otros lenguajes en Lambda, como [Go](#) o [Rust](#), utilice un [tiempo de ejecución exclusivo del sistema operativo](#). El entorno de ejecución de Lambda proporciona una [interfaz de tiempo de ejecución](#) para obtener eventos de invocación y enviar respuestas. Puede implementar otros lenguajes mediante un [tiempo de ejecución personalizado](#) junto con su código de función, o en una [capa](#).

## Tiempos de ejecución admitidos


En la siguiente tabla se enumeran los tiempos de ejecución de Lambda admitidos y las fechas de caducidad proyectadas. Incluso cuando un tiempo de ejecución está obsoleto, puede seguir

creando y actualizando funciones durante un período limitado. Para obtener más información, consulte [the section called “Uso del tiempo de ejecución después de la obsolescencia”](#). En la tabla se muestran las fechas previstas actualmente para la caducidad del tiempo de ejecución. Estas fechas se proporcionan para fines de planificación y están sujetas a cambios.

### Tiempos de ejecución admitidos

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Node.js 20	nodejs20.x	Amazon Linux 2023			
Node.js 18	nodejs18.x	Amazon Linux 2			
Python 3.12	python3.12	Amazon Linux 2023			
Python 3.11	python3.11	Amazon Linux 2			
Python 3.10	python3.10	Amazon Linux 2			
Python 3.9	python3.9	Amazon Linux 2			
Python 3.8	python3.8	Amazon Linux 2	14 de octubre de 2024	28 de febrero de 2025	31 de marzo de 2025
Java 21	java21	Amazon Linux 2023			
Java 17	java17	Amazon Linux 2			
Java 11	java11	Amazon Linux 2			

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Java 8	java8.a12	Amazon Linux 2			
.NET 8	dotnet8	Amazon Linux 2023			
.NET 6	dotnet6	Amazon Linux 2	20 de diciembre de 2024	28 de febrero de 2025	31 de marzo de 2025
Ruby 3.3	ruby3.3	Amazon Linux 2023			
Ruby 3.2	ruby3.2	Amazon Linux 2			
Tiempo de ejecución exclusivo del sistema operativo	provided.a12023	Amazon Linux 2023			
Tiempo de ejecución exclusivo del sistema operativo	provided.a12	Amazon Linux 2			

 Note

Para las nuevas regiones, Lambda no admitirá tiempos de ejecución que queden obsoletos dentro de los seis meses posteriores.

Lambda mantiene los tiempos de ejecución administrados y sus imágenes de base de contenedor correspondientes actualizados con parches y la compatibilidad con lanzamientos de versiones menores. Para obtener más información, consulte [Actualizaciones de tiempo de ejecución de Lambda](#).

Lambda sigue siendo compatible con el lenguaje de programación Go tras la caducidad del tiempo de ejecución de Go 1.x. Para obtener más información, consulte [Migración de las funciones de AWS Lambda del tiempo de ejecución de Go1.x al tiempo de ejecución personalizado en Amazon Linux 2](#) en el Blog de informática de AWS.

Todos los tiempos de ejecución compatibles de Lambda admiten tanto arquitecturas x86\_64 como arm64.

## Nuevas versiones de tiempo de ejecución

Para las nuevas versiones de los lenguajes, Lambda solo proporciona tiempos de ejecución administrados cuando la versión llega a la fase de soporte a largo plazo (LTS) del ciclo de lanzamiento del lenguaje. Por ejemplo, para el [ciclo de lanzamiento de Node.js](#), cuando la versión llega a la fase de LTS activa.

Antes de que la versión alcance la fase de soporte a largo plazo, permanece en desarrollo y puede estar sujeta a cambios importantes. Lambda aplica las actualizaciones del tiempo de ejecución automáticamente de forma predeterminada, por lo que interrumpir los cambios en una versión del tiempo de ejecución podría alterar el funcionamiento esperado de las funciones.

Lambda no proporciona tiempos de ejecución administrados para las versiones de lenguajes que no tendrán una versión de LTS.

La siguiente lista muestra el mes de lanzamiento previsto para los próximos tiempos de ejecución de Lambda. Estas fechas son solo indicativas y están sujetas a cambios.

- Python 3.13: noviembre de 2024
- Node.js 22: noviembre de 2024

## Política de obsolescencia del tiempo de ejecución

[Tiempos de ejecución de Lambda](#) para archivos .zip se crean en torno a una combinación de bibliotecas de sistemas operativos, lenguaje de programación y software que están sujetos a actualizaciones de mantenimiento y seguridad. La política de obsolescencia estándar de Lambda



consiste en dejar obsoleto un tiempo de ejecución cuando algún componente importante de este llegue al final del soporte comunitario a largo plazo (LTS) y las actualizaciones de seguridad ya no estén disponibles. Por lo general, este es el tiempo de ejecución del lenguaje, aunque en algunos casos, un tiempo de ejecución puede quedar obsoleto porque el sistema operativo (SO) llega al final del LTS.

Cuando un tiempo de ejecución queda obsoleto, AWS puede dejar de aplicar revisiones de seguridad o actualizaciones a ese tiempo de ejecución y las funciones que utilizan ese tiempo de ejecución dejan de ser aptas para recibir asistencia técnica. Estos tiempos de ejecución obsoletos se proporcionan “tal cual”, sin garantía alguna, y pueden contener errores, defectos u otras vulnerabilidades.

Para obtener más información sobre la administración de las actualizaciones y la caducidad del tiempo de ejecución, consulte las próximas secciones y el artículo [Administración de las actualizaciones de los tiempos de ejecución de AWS Lambda](#) en el Blog de informática de AWS.

#### Important

A veces Lambda retrasa la obsolescencia de un tiempo de ejecución de Lambda durante un periodo limitado luego de la fecha de finalización del soporte de la versión del lenguaje compatible con el tiempo de ejecución. Durante este periodo, Lambda solo aplica parches de seguridad al sistema operativo del tiempo de ejecución. Lambda no aplica parches de seguridad a los tiempos de ejecución de los lenguajes de programación una vez que llegan a la fecha de finalización del soporte.

## Modelo de responsabilidad compartida

Lambda es responsable de seleccionar y publicar las actualizaciones de seguridad para todos los tiempos de ejecución administrados y las imágenes base de contenedores compatibles. De forma predeterminada, Lambda aplicará estas actualizaciones a las funciones de forma automática mediante tiempos de ejecución administrados. Si se ha modificado la configuración predeterminada de actualización automática del tiempo de ejecución, consulte el [modelo de responsabilidad compartida de los controles de administración del tiempo de ejecución](#). En el caso de las funciones implementadas mediante imágenes de contenedor, es responsable de volver a crear la imagen del contenedor de la función a partir de la imagen base más reciente y volver a implementar la imagen del contenedor.

Cuando un tiempo de ejecución queda obsoleto, Lambda deja de ser responsable de actualizar el tiempo de ejecución administrado y las imágenes base del contenedor. Es responsable de actualizar sus funciones para utilizar un tiempo de ejecución o una imagen base compatibles.

En todos los casos, es responsable de aplicar las actualizaciones al código de la función, incluidas las de sus dependencias. Sus responsabilidades en virtud del modelo de responsabilidad compartida se resumen en la siguiente tabla.

Fase del ciclo de vida del tiempo de ejecución	Responsabilidades de Lambda	Sus responsabilidades
Tiempo de ejecución administrado compatible	<p>Proporcionar actualizaciones periódicas del tiempo de ejecución con revisiones de seguridad y otras actualizaciones.</p> <p>Aplicar las actualizaciones del tiempo de ejecución automáticamente de forma predeterminada (consulte los comportamientos no predeterminados en <a href="#">the section called “Modos de actualización del tiempo de ejecución”</a>).</p>	<p>Actualizar el código de la función, incluidas las dependencias, para corregir cualquier vulnerabilidad de seguridad.</p>
Imagen de contenedor compatible	<p>Proporcionar actualizaciones periódicas en la imagen base del contenedor con revisiones de seguridad y otras actualizaciones.</p>	<p>Actualizar el código de la función, incluidas las dependencias, para corregir cualquier vulnerabilidad de seguridad.</p> <p>Volver a compilar e implementar la imagen del contenedor con regularidad utilizando la imagen base más reciente.</p>

Fase del ciclo de vida del tiempo de ejecución	Responsabilidades de Lambda	Sus responsabilidades
El tiempo de ejecución administrado está a punto de quedar obsoleto	<p>Notificar a los clientes antes de que el tiempo de ejecución quede obsoleto mediante la documentación, AWS Health Dashboard, el correo electrónico y Trusted Advisor.</p> <p>La responsabilidad de las actualizaciones del tiempo de ejecución finaliza cuando queda obsoleto.</p>	<p>Supervisar la documentación de Lambda, AWS Health Dashboard, el correo electrónico o Trusted Advisor para consultar la información sobre la obsolescencia del tiempo de ejecución.</p> <p>Actualizar las funciones a un tiempo de ejecución compatible antes de que el tiempo de ejecución anterior quede obsoleto.</p>
La imagen del contenedor está a punto de quedar obsoleta	<p>Las notificaciones de obsolescencia no están disponibles para las funciones que utilizan imágenes de contenedores.</p> <p>La responsabilidad de las actualizaciones de la imagen base del contenedor finaliza cuando queda obsoleta.</p>	Tenga en cuenta los programas de obsolescencia y actualice las funciones a una imagen base compatible antes de que la imagen anterior quede obsoleta.

## Uso del tiempo de ejecución después de la obsolescencia

Cuando un tiempo de ejecución queda obsoleto, AWS puede dejar de aplicar revisiones de seguridad o actualizaciones a ese tiempo de ejecución y las funciones que utilizan ese tiempo de ejecución dejan de ser aptas para recibir asistencia técnica. Estos tiempos de ejecución obsoletos se proporcionan “tal cual”, sin garantía alguna, y pueden contener errores, defectos u otras vulnerabilidades. Las funciones que utilizan un tiempo de ejecución obsoleto también pueden tener un rendimiento inferior u otros problemas, como la caducidad del certificado, lo que puede provocar que no funcionen de forma correcta.

Podrá seguir creando nuevas funciones de Lambda con un tiempo de ejecución durante al menos 30 días luego de que ese tiempo de ejecución caduque. A partir de los 30 días posteriores a la caducidad, Lambda comienza a bloquear la creación de funciones nuevas.

Podrá seguir actualizando el código y la configuración de las funciones existentes durante al menos 60 días desde la obsolescencia de ese tiempo de ejecución. A partir de los 60 días posteriores a la obsolescencia, Lambda comenzará a bloquear la actualización del código y la configuración de las funciones existentes.

#### Note

Para algunos tiempos de ejecución, AWS está retrasando las fechas de creación y de actualización de la función de bloqueo a más de los 30 y 60 días habituales posteriores a la obsolescencia. AWS ha realizado este cambio en respuesta a los comentarios de los clientes para que se disponga de más tiempo para actualizar sus funciones. Consulte las tablas en [the section called “Tiempos de ejecución admitidos”](#) y [the section called “Tiempos de ejecución obsoletos”](#) para poder ver las fechas de su tiempo de ejecución.

Puede actualizar una función para usar una versión más reciente del tiempo de ejecución compatible de forma indefinida luego de que un tiempo de ejecución quede obsoleto. Debe comprobar que la función funcione con el nuevo tiempo de ejecución antes de aplicar el cambio de tiempo de ejecución en los entornos de producción, ya que no podrá volver al tiempo de ejecución obsoleto una vez transcurrido el periodo de 60 días. Se recomienda utilizar [versiones](#) y [alias](#) de las funciones para permitir una implementación segura con reversión.

Tenga en cuenta que el tiempo exacto durante el que puede seguir creando y actualizando funciones no es fijo. Este período puede variar en función de la caducidad y de cada Regiones de AWS. Las fechas nominales para el bloqueo de la creación y actualización de funciones se indican en la tabla de tiempos de ejecución compatibles que se encuentra la primera sección de esta página. Lambda no comenzará a bloquear la creación o actualización de funciones antes de las fechas que se indican en esta tabla.

Puede seguir invocando sus funciones indefinidamente una vez que el tiempo de ejecución caduque. Sin embargo, AWS recomienda enfáticamente migrar las funciones a un tiempo de ejecución compatible para que continúe recibiendo parches de seguridad y siga siendo elegible para soporte técnico.

# Recepción de notificaciones de caducidad de un tiempo de ejecución

Cuando un tiempo de ejecución se acerca a su fecha de caducidad, Lambda le envía una alerta por correo electrónico si alguna de las funciones de su Cuenta de AWS utiliza ese tiempo de ejecución. Las notificaciones también se muestran en AWS Health Dashboard y en AWS Trusted Advisor.

- Recepción de notificaciones por correo electrónico:

Lambda le envía una alerta por correo electrónico al menos 180 días antes de que el tiempo de ejecución caduque. En este correo electrónico se enumeran las versiones \$LATEST de todas las funciones que utilizan el tiempo de ejecución. Para ver una lista completa de las versiones de funciones afectadas, utilice Trusted Advisor o consulte [the section called “Obtener datos sobre las funciones por tiempo de ejecución”](#).

Lambda envía una notificación por correo electrónico al contacto de su cuenta principal de Cuenta de AWS. Para obtener información sobre cómo ver o actualizar las direcciones de correo electrónico de su cuenta, consulte [Actualización de la información de contacto](#) en la Referencia general de AWS.

- Recepción de notificaciones a través del AWS Health Dashboard:

El AWS Health Dashboard muestra una notificación al menos 180 días antes de que el tiempo de ejecución caduque. Las notificaciones aparecen en la página de Estado de cuenta, en [Otras notificaciones](#). En la pestaña Recursos afectados de la notificación se enumeran las versiones \$LATEST de todas las funciones que utilizan el tiempo de ejecución.

## Note

Para ver una lista completa de las versiones de funciones afectadas, utilice Trusted Advisor o consulte [the section called “Obtener datos sobre las funciones por tiempo de ejecución”](#).

Las notificaciones de AWS Health Dashboard caducan 90 días luego de que el tiempo de ejecución afectado caduque.

- Uso de AWS Trusted Advisor

Trusted Advisor muestra una notificación al menos 180 días antes de que el tiempo de ejecución caduque. Las notificaciones aparecen en la página [Seguridad](#). En Funciones de AWS Lambda que utilizan tiempos de ejecución obsoletos se muestra una lista las funciones afectadas. Esta lista de funciones muestra tanto las versiones \$LATEST como las publicadas y se actualiza automáticamente para reflejar el estado actual de las funciones.

Puede activar las notificaciones semanales por correo electrónico desde Trusted Advisor, en la página [Preferencias](#) de la consola de Trusted Advisor.

## Tiempos de ejecución obsoletos

Los siguientes tiempos de ejecución han quedado obsoletos:

### Tiempos de ejecución obsoletos

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Node.js 16	nodejs16.x	Amazon Linux 2	12 de junio de 2024	28 de febrero de 2025	31 de marzo de 2025
.NET 7 (solo contenedor)	dotnet7	Amazon Linux 2	14 de mayo de 2024		
Java 8	java8	Amazon Linux	8 de enero de 2024	8 de febrero de 2024	28 de febrero de 2025
Go 1.x	go1.x	Amazon Linux	8 de enero de 2024	8 de febrero de 2024	28 de febrero de 2025
Tiempo de ejecución exclusivo del sistema operativo	provided	Amazon Linux	8 de enero de 2024	8 de febrero de 2024	28 de febrero de 2025

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Ruby 2.7	ruby2.7	Amazon Linux 2	7 de diciembre de 2023	9 de enero de 2024	28 de febrero de 2025
Node.js 14	nodejs14.x	Amazon Linux 2	4 de diciembre de 2023	9 de enero de 2024	28 de febrero de 2025
Python 3.7	python3.7	Amazon Linux	4 de diciembre de 2023	9 de enero de 2024	28 de febrero de 2025
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	3 de abril de 2023	3 de abril de 2023	3 de mayo de 2023
Node.js 12	nodejs12.x	Amazon Linux 2	31 de marzo de 2023	31 de marzo de 2023	30 de abril de 2023
Python 3.6	python3.6	Amazon Linux	18 de julio de 2022	18 de julio de 2022	29 de agosto de 2022
.NET 5 (solo contenedor)	dotnet5.0	Amazon Linux 2	10 de mayo de 2022		
.NET Core 2.1	dotnetcore2.1	Amazon Linux	5 de enero de 2022	5 de enero de 2022	13 de abril de 2022
Node.js 10	nodejs10.x	Amazon Linux 2	30 de julio de 2021	30 de julio de 2021	14 de febrero de 2022
Ruby 2.5	ruby2.5	Amazon Linux	30 de julio de 2021	30 de julio de 2021	31 de marzo de 2022
Python 2.7	python2.7	Amazon Linux	15 de julio de 2021	15 de julio de 2021	30 de mayo de 2022

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Node.js 8.10	nodejs8.10	Amazon Linux	6 de marzo de 2020		6 de marzo de 2020
Node.js 4.3	nodejs4.3	Amazon Linux	5 de marzo de 2020		5 de marzo de 2020
Node.js 4.3 edge	nodejs4.3-edge	Amazon Linux	5 de marzo de 2020		30 de abril de 2019
Node.js 6.10	nodejs6.10	Amazon Linux	12 de agosto de 2019	12 de agosto de 2019	
.NET Core 1.0	dotnetcore1.0	Amazon Linux	27 de junio de 2019		30 de julio de 2019
.NET Core 2.0	dotnetcore2.0	Amazon Linux	30 de mayo de 2019		30 de mayo de 2019
Node.js 0.10	nodejs	Amazon Linux			31 de octubre de 2016

En casi todos los casos, la fecha de fin de vida útil de una versión de un lenguaje o un sistema operativo se conoce con mucha antelación. Los enlaces que aparecen a continuación indican los cronogramas de fin de vida útil de cada lenguaje que Lambda admite como tiempo de ejecución administrado.

#### Políticas de soporte de lenguajes y marcos

- Node.js: [github.com](https://github.com)
- Python: [devguide.python.org](https://devguide.python.org)
- Ruby: [www.ruby-lang.org](https://www.ruby-lang.org)
- Java: [www.oracle.com](https://www.oracle.com) y [Preguntas frecuentes de Corretto](#)
- Go: [golang.org](https://golang.org)



- .NET: [dotnet.microsoft.com](https://dotnet.microsoft.com)

# Cómo entender la forma en que Lambda administra las actualizaciones de las versiones de tiempo de ejecución

Lambda mantiene todos los tiempos de ejecución gestionados al día con actualizaciones de seguridad, correcciones de errores, nuevas funciones, mejoras de rendimiento y soporte para versiones menores. Estas actualizaciones del tiempo de ejecución se publican como versiones del tiempo de ejecución. Lambda aplica las actualizaciones del tiempo de ejecución a las funciones mediante la migración de la función de una versión de tiempo de ejecución anterior a una nueva versión del tiempo de ejecución.

De forma predeterminada, para las funciones que utilizan tiempos de ejecución administrados, Lambda aplica las actualizaciones del tiempo de ejecución de forma automática. Con las actualizaciones automáticas en tiempo de ejecución, Lambda asume la carga operativa de aplicar parches a las versiones en tiempo de ejecución. Para la mayoría de los clientes, las actualizaciones automáticas son la opción correcta. Puede cambiar este comportamiento predeterminado [configurando los ajustes de administración del tiempo de ejecución](#).

Lambda también publica cada nueva versión en tiempo de ejecución como una imagen de contenedor. Para actualizar las versiones del tiempo de ejecución de las funciones basadas en contenedores, debe [crear una nueva imagen de contenedor](#) a partir de la imagen base actualizada y volver a implementar la función.

Cada versión en tiempo de ejecución está asociada a un número de versión y un ARN (Amazon Resource Name). Los números de versión del tiempo de ejecución utilizan un esquema de numeración definido por Lambda, de forma independiente de los números de versión que utiliza el lenguaje de programación. El ARN de la versión en tiempo de ejecución es un identificador único para cada versión del tiempo de ejecución. Puede ver el ARN de la versión del tiempo de ejecución actual de la función en la consola de Lambda y en la [línea INIT\\_START de los registros de la función](#).

Las versiones del tiempo de ejecución no deben confundirse con los identificadores de tiempo de ejecución. Cada tiempo de ejecución tiene un identificador de tiempo de ejecución único, como `python3.12` o `nodejs20.x`. Se corresponden con cada versión principal del lenguaje de programación. Las versiones del tiempo de ejecución describen la versión de parche de un entorno de ejecución individual.

**Note**

El ARN para el mismo número de versión de tiempo de ejecución puede variar entre las arquitecturas de CPU y Regiones de AWS.

## Temas

- [Modos de actualización del tiempo de ejecución](#)
- [Lanzamiento de la versión del tiempo de ejecución bifásico](#)
- [Configuración de los ajustes de administración del tiempo de ejecución de Lambda](#)
- [Restauración de una versión del tiempo de ejecución de Lambda](#)
- [Identificación de los cambios de versión del tiempo de ejecución de Lambda](#)
- [Cómo entender el modelo de responsabilidad compartida para la administración del tiempo de ejecución de Lambda](#)
- [Control de los permisos de actualización del tiempo de ejecución de Lambda para aplicaciones de alta conformidad](#)

## Modos de actualización del tiempo de ejecución

Lambda se esfuerza por proporcionar actualizaciones de tiempo de ejecución que sean compatibles con versiones anteriores a las funciones existentes. Sin embargo, al igual que ocurre con los parches de software, hay casos excepcionales en los que una actualización del tiempo de ejecución puede afectar negativamente a una función ya existente. Por ejemplo, los parches de seguridad pueden exponer un problema subyacente con una función existente que depende del comportamiento inseguro anterior. Los controles de administración del tiempo de ejecución de Lambda ayudan a reducir el riesgo de que las cargas de trabajo se vean afectadas en el caso de que se produzca una incompatibilidad entre las versiones en tiempo de ejecución. Para cada [versión de función](#) (\$LATEST o versión publicada), puede elegir uno de los siguientes modos de actualización en tiempo de ejecución:

- Auto(predeterminado): actualice de forma automática la versión de ejecución más reciente y segura mediante [Lanzamiento de la versión del tiempo de ejecución bifásico](#). Recomendamos este modo a la mayoría de los clientes para que siempre se beneficien de las actualizaciones del tiempo de ejecución.

- **Actualización de función:** actualiza a la versión del tiempo de ejecución más reciente y segura cuando actualice su función. Cuando actualiza la función, Lambda actualiza el tiempo de ejecución de la función a la versión de ejecución más reciente y segura. Este enfoque sincroniza las actualizaciones en tiempo de ejecución con las implementaciones de funciones, lo que le permite controlar cuándo Lambda aplica las actualizaciones en tiempo de ejecución. Con este modo, puede detectar y mitigar con anticipación las incompatibilidades poco frecuentes entre las actualizaciones en tiempo de ejecución. Al utilizar este modo, debe actualizar periódicamente las funciones para mantener su tiempo de ejecución actualizado.
- **Manual:** actualice de forma manual la versión del tiempo de ejecución. Se especifica una versión del tiempo de ejecución en la configuración de la función. La función utiliza esta versión del tiempo de ejecución de forma indefinida. En el caso de que una nueva versión del tiempo de ejecución sea incompatible con una función existente, puede utilizar este modo para restaurar la función a una versión anterior en tiempo de ejecución. Recomendamos no utilizar el modo Manual para intentar lograr la coherencia del tiempo de ejecución en todas las implementaciones. Para obtener más información, consulte [Restauración de una versión del tiempo de ejecución de Lambda](#).

La responsabilidad de aplicar las actualizaciones de tiempo de ejecución a las funciones varía según el modo de actualización del tiempo de ejecución que elija. Para obtener más información, consulte [Cómo entender el modelo de responsabilidad compartida para la administración del tiempo de ejecución de Lambda](#).

## Lanzamiento de la versión del tiempo de ejecución bifásico

Lambda presenta nuevas versiones del tiempo de ejecución en el siguiente orden:

1. En la primera fase, Lambda aplica la nueva versión del tiempo de ejecución cada vez que se crea o se actualiza una función. La función se actualiza al llamar a las operaciones de la API [updateFunctionCode](#) o [UpdateFunctionConfiguration](#).
2. En la segunda fase, Lambda actualiza cualquier función que utilice el modo de actualización Auto (Automático) del tiempo de ejecución y que aún no se haya actualizado a la nueva versión del tiempo de ejecución.

La duración total del proceso de implementación varía en función de varios factores, incluida la gravedad de los parches de seguridad incluidos en la actualización del tiempo de ejecución.

Si está desarrollando e implementando sus funciones de forma activa, lo más probable es que aprendas nuevas versiones del tiempo de ejecución durante la primera fase. Esto sincroniza las

actualizaciones del tiempo de ejecución con las actualizaciones de funciones. En el caso excepcional de que la última versión del tiempo de ejecución afecte de forma negativa a la aplicación, este enfoque le permite tomar medidas correctivas inmediatas. Las funciones que no están en desarrollo activo siguen recibiendo la ventaja operativa de las actualizaciones automáticas del tiempo de ejecución durante la segunda fase.

Este enfoque no afecta a las funciones configuradas en modo Function update (Actualización de funciones) o Manual. Las funciones que utilizan el modo Function update (Actualización de funciones) reciben las actualizaciones de ejecución más recientes solo cuando se crean o actualizan. Las funciones que utilizan el modo Manual no reciben actualizaciones en tiempo de ejecución.

Lambda publica nuevas versiones en tiempo de ejecución de forma gradual a través de las Regiones de AWS. Si tus funciones están configuradas en modo Function update (Actualización de funciones) o Auto (Automático), es posible que estas se desplieguen al mismo tiempo en diferentes regiones, o en diferentes momentos de la misma región, elijan versiones de ejecución diferentes. Los clientes que necesiten garantizar la coherencia de las versiones del tiempo de ejecución en sus entornos deben [utilizar imágenes de contenedores para implementar sus funciones de Lambda](#). El modo Manual está diseñado como una mitigación temporal para permitir la reversión de la versión del tiempo de ejecución en el caso excepcional de que una versión del tiempo de ejecución sea incompatible con la función.

## Configuración de los ajustes de administración del tiempo de ejecución de Lambda

Puede configurar los ajustes de administración del tiempo de ejecución mediante la consola de Lambda o AWS Command Line Interface (AWS CLI).

### Note

Puede configurar los ajustes de administración del tiempo de ejecución por separado para cada [versión de la función](#).

### Configuración de la versión de tiempo de ejecución de Lambda (consola)

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.

3. En la pestaña Code (Código), en Runtime settings (Configuración del tiempo de ejecución), seleccione Edit runtime management configuration (Editar la configuración de administración del tiempo de ejecución).
4. En Runtime management configuration (Configuración de la administración del tiempo de ejecución), elija una de las siguientes opciones:
  - Para que la función se actualice automáticamente a la última versión de ejecución, seleccione Auto (Automático).
  - Para que la función se actualice a la última versión del tiempo de ejecución al cambiarla, seleccione Function update (Actualización de función).
  - Para que la función se actualice a la última versión del tiempo de ejecución solo cuando cambie el ARN de la versión en tiempo de ejecución, seleccione Manual. Encontrará el ARN de la versión de tiempo de ejecución en Runtime management configuration (Configuración de la administración del tiempo de ejecución). También puede encontrar el ARN en la línea INIT\_START de sus registros de funciones.

Para obtener más información sobre estas opciones, consulte [Modos de actualización en tiempo de ejecución](#).

5. Seleccione Save (Guardar).

Para configurar cómo Lambda actualiza su versión del tiempo de ejecución (AWS CLI)

Para configurar la administración del tiempo de ejecución de una función, ejecute el comando de AWS CLI [put-runtime-management-config](#). Cuando utilice el modo Manual, también debe proporcionar el ARN de la versión en tiempo de ejecución.

```
aws lambda put-runtime-management-config \  
  --function-name my-function \  
  --update-runtime-on Manual \  
  --runtime-version-arn arn:aws:lambda:us-east-2:runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1
```

Debería ver una salida similar a esta:

```
{  
  "UpdateRuntimeOn": "Manual",  
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
```

```
"RuntimeVersionArn": "arn:aws:lambda:us-east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
}
```

## Restauración de una versión del tiempo de ejecución de Lambda

En el caso de que una nueva versión del tiempo de ejecución sea incompatible con la función existente, puede restaurar su versión en tiempo de ejecución a una anterior. Esto mantiene la aplicación en funcionamiento y minimiza las interrupciones, lo que proporciona tiempo para corregir la incompatibilidad antes de volver a la última versión del tiempo de ejecución.

Lambda no impone un límite de tiempo para el uso de una versión de ejecución determinada. Sin embargo, le recomendamos actualizar a la última versión del tiempo de ejecución lo antes posible para disfrutar de los parches de seguridad, las mejoras de rendimiento y las funciones más recientes. Lambda ofrece la opción de restaurar a una versión anterior del tiempo de ejecución únicamente como medida de mitigación temporal en el caso de que surja un problema de compatibilidad con las actualizaciones del tiempo de ejecución. Las funciones que utilizan una versión anterior del tiempo de ejecución durante un periodo prolongado pueden presentar problemas o tener un rendimiento inferior, como la caducidad del certificado, lo que puede provocar que no funcionen de forma correcta.

Puede restaurar a una versión del tiempo de ejecución de las siguientes maneras:

- [Uso de modo de actualización Manual en tiempo de ejecución](#)
- [Uso de versiones de funciones publicadas](#)

Para obtener más información, consulte [Introducción a los controles de administración de los tiempos de ejecución de AWS Lambda](#) en el Blog de computación de AWS.

### Restaurar una versión del tiempo de ejecución mediante el modo de actualización Manual

Si utiliza el modo de actualización Auto (Automático) de la versión del tiempo de ejecución o utiliza la versión \$LATEST del tiempo de ejecución, puede restaurar su versión del tiempo de ejecución mediante el modo Manual. Para la [versión de la función](#) que desea restaurar, cambie el modo de actualización de la versión del tiempo de ejecución a Manual y especifique el ARN de la versión del tiempo de ejecución anterior. Para obtener más información sobre cómo encontrar el ARN de la

versión anterior del tiempo de ejecución, consulte [Identificación de los cambios de versión del tiempo de ejecución de Lambda](#).

#### Note

Si la versión \$LATEST de la función está configurada para usar el modo Manual, no puede cambiar la arquitectura del CPU ni la versión de tiempo de ejecución que usa la función. Para realizar estos cambios, debe cambiar al modo Auto (Automático) o Function update (Actualización de funciones).

## Deshacer una versión en tiempo de ejecución por medio de las versiones de funciones publicadas

Las [versiones de funciones](#) publicadas son una instantánea inmutable del código y la configuración de la función \$LATEST en el momento en que se crearon. En el modo Auto (Automático), Lambda actualiza de forma automática la versión del tiempo de ejecución de las versiones de funciones publicadas durante la segunda fase del despliegue de la versión del tiempo de ejecución. En el modo de Function update (Actualización de funciones), Lambda no actualiza la versión del tiempo de ejecución de las versiones de funciones publicadas.

Por lo tanto, las versiones de funciones publicadas mediante el modo Function update (Actualización de funciones) crean una instantánea estática del código de la función, la configuración y la versión del tiempo de ejecución. Al utilizar el modo Function update (Actualización de funciones) con las versiones de las funciones, puede sincronizar las actualizaciones del tiempo de ejecución con sus implementaciones. También puede coordinar la reversión de las versiones de código, configuración y tiempo de ejecución redirigiendo el tráfico a una versión de función publicada anteriormente. Puede integrar este enfoque en su sistema de integración y entrega continuas (CI/CD) para lograr una reversión completamente automática en el caso de que se produzca una incompatibilidad entre las actualizaciones del tiempo de ejecución. Al utilizar este enfoque, debe actualizar la función con regularidad y publicar nuevas versiones de la función para obtener las actualizaciones más recientes del tiempo de ejecución. Para obtener más información, consulte [Cómo entender el modelo de responsabilidad compartida para la administración del tiempo de ejecución de Lambda](#).

## Identificación de los cambios de versión del tiempo de ejecución de Lambda

El número de versión del tiempo de ejecución y el ARN se registran en la línea de registro INIT\_START, que Lambda emite a los registros de CloudWatch cada vez que crea un nuevo [entorno](#)



[de ejecución](#). Dado que el entorno de ejecución utiliza la misma versión de tiempo de ejecución para todas las invocaciones de funciones, Lambda emite la línea de registro INIT\_START solo cuando ejecuta la fase inicial. Lambda no emite esta línea de registro para cada invocación de función. Lambda emite la línea de registro a los Registros de CloudWatch, pero no aparece visible en la consola.

### Example Ejemplo de línea de registro INIT\_START

```
INIT_START Runtime Version: python:3.9.v14    Runtime Version ARN: arn:aws:lambda:eu-south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

En lugar de trabajar directamente con los registros, puede utilizar [Información de colaboradores de Amazon CloudWatch](#) para identificar las transiciones entre las versiones en tiempo de ejecución. La siguiente regla cuenta las distintas versiones del tiempo de ejecución de cada línea de registro INIT\_START. Para usar la regla, sustituya el nombre del grupo de registro de ejemplo /aws/Lambda/\* por el prefijo adecuado para su función o grupo de funciones.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "eventType",
        "In": [
          "INIT_START"
        ]
      }
    ],
    "Keys": [
      "runtimeVersion",
      "runtimeVersionArn"
    ]
  },
  "LogFormat": "CLF",
  "LogGroupNames": [
    "/aws/Lambda/*"
  ],
}
```

```

"Fields": {
  "1": "eventType",
  "4": "runtimeVersion",
  "8": "runtimeVersionArn"
}
}

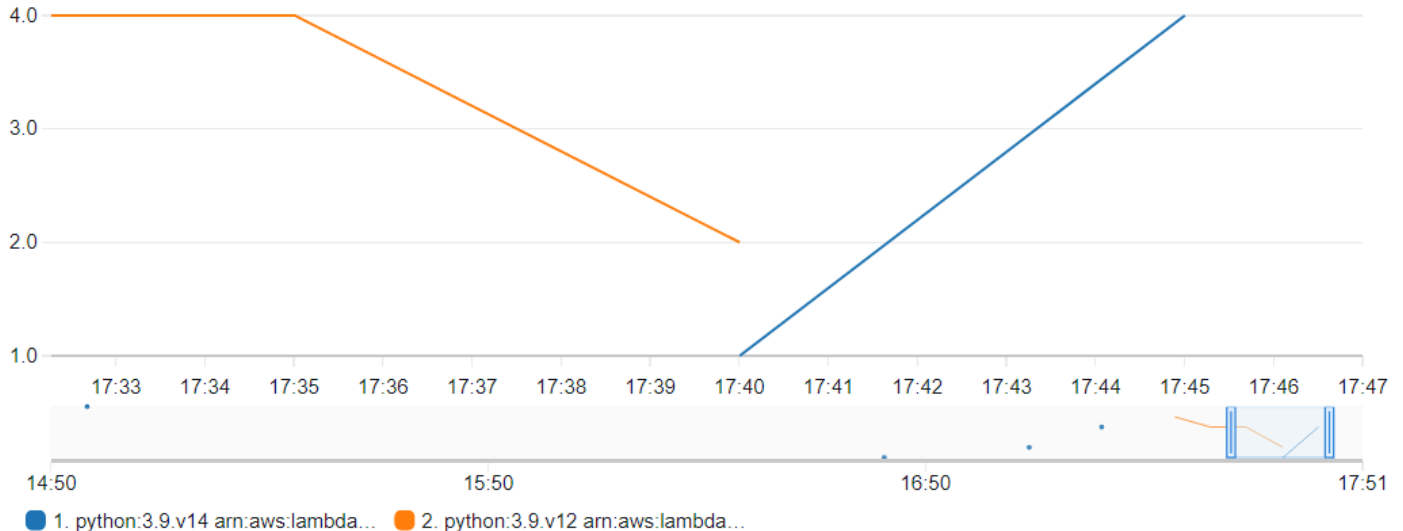
```

El siguiente informe de Información de colaboradores de CloudWatch muestra un ejemplo de una transición de versión del tiempo de ejecución tal como se recoge en la regla anterior. La línea naranja muestra la inicialización del entorno de ejecución para la versión anterior del tiempo de ejecución (python:3.9.v12) y la línea azul muestra la inicialización del entorno de ejecución para la nueva versión del tiempo de ejecución (python:3.9.v14).

Top 2 of 2 unique contributors



2 unique contributors • No unit



## Cómo entender el modelo de responsabilidad compartida para la administración del tiempo de ejecución de Lambda

Lambda es responsable de seleccionar y publicar las actualizaciones de seguridad para todos los tiempos de ejecución administrados y las imágenes de contenedores compatibles. La responsabilidad de actualizar las funciones existentes para utilizar la última versión del tiempo de ejecución varía según el modo de actualización en tiempo de ejecución que utilice.

Lambda es responsable de aplicar las actualizaciones del tiempo de ejecución a todas las funciones configuradas para usar el modo de actualización Auto (Automático) del tiempo de ejecución.

En el caso de las funciones configuradas con el modo de actualización del tiempo de ejecución Function update (Actualización de función), usted es responsable de actualizar regularmente su función. Lambda es responsable de aplicar las actualizaciones del tiempo de ejecución al realizar dichas actualizaciones. Si no actualiza la función, Lambda no actualizará el tiempo de ejecución. Si no actualiza la función con regularidad, le recomendamos que la configure para que se actualice de forma automática en tiempo de ejecución para que siga recibiendo actualizaciones de seguridad.

En el caso de las funciones configuradas para usar el modo de actualización Manual en tiempo de ejecución, usted es responsable de actualizar la función para que utilice la versión más reciente en tiempo de ejecución. Le recomendamos encarecidamente que utilice este modo solo para revertir la versión en tiempo de ejecución como medida de mitigación temporal en el raro caso de que se produzca una incompatibilidad entre las actualizaciones en tiempo de ejecución. También le recomendamos que cambie al modo Auto (Automático) lo antes posible para minimizar el tiempo en el que las funciones no se parchean.

Si [utiliza imágenes de contenedores para implementar sus funciones](#), Lambda es responsable de publicar las imágenes base actualizadas. En este caso, es responsable de reconstruir la imagen del contenedor de la función a partir de la imagen de base más reciente y volver a implementar la imagen del contenedor.

Esto se resume en la siguiente tabla:

Modo de implementación	Responsabilidad de Lambda	Responsabilidades del cliente
Tiempo de ejecución gestionado, modo Auto (Automático)	<p>Publique nuevas versiones en tiempo de ejecución que contengan los parches más recientes.</p> <p>Aplique parches de ejecución a las funciones existentes.</p>	Vuelva a una versión anterior en tiempo de ejecución en el caso de que surja un problema de compatibilidad con las actualizaciones en tiempo de ejecución.
Tiempo de ejecución gestionado, modo Function	Publique nuevas versiones en tiempo de ejecución que contengan los parches más recientes.	Actualice las funciones con regularidad para obtener la última versión en tiempo de ejecución.

Modo de implementación	Responsabilidad de Lambda	Responsabilidades del cliente
update (Actualización de funciones)		<p>Cambie una función a modo Auto (Automático) cuando no la actualice de forma regular.</p> <p>Vuelva a una versión anterior en tiempo de ejecución en el caso de que surja un problema de compatibilidad con las actualizaciones en tiempo de ejecución.</p>
Tiempo de ejecución gestionado, modo Manual	<p>Publique nuevas versiones en tiempo de ejecución que contengan los parches más recientes.</p>	<p>Utilice este modo solo para revertir temporalmente el tiempo de ejecución en el caso de que surja un problema de compatibilidad con las actualizaciones en tiempo de ejecución.</p> <p>Cambie las funciones al modo Auto (Automático) o Function update (Actualización de funciones) y a la última versión de ejecución lo antes posible.</p>
Imagen de contenedor	<p>Publique nuevas imágenes de contenedores que contengan los parches más recientes.</p>	<p>Vuelva a implementar las funciones con regularidad utilizando la imagen base del contenedor más reciente para obtener los parches más recientes.</p>

Para obtener más información sobre la responsabilidad compartida con AWS, consulte [Modelo de responsabilidad compartida](#).

## Control de los permisos de actualización del tiempo de ejecución de Lambda para aplicaciones de alta conformidad

Para cumplir con los requisitos de parches, los clientes de Lambda suelen confiar en las actualizaciones automáticas en tiempo de ejecución. Si su aplicación está sujeta a estrictos requisitos de actualización de parches, es posible que desee limitar el uso de versiones anteriores en tiempo de ejecución. Puede restringir los controles de administración del tiempo de ejecución de Lambda mediante AWS Identity and Access Management (IAM) para denegar a los usuarios de

su cuenta AWS el acceso a la operación de la API [PutRuntimeManagementConfig](#). Esta operación se utiliza para elegir el modo de actualización en tiempo de ejecución de una función. Al denegar el acceso a esta operación, todas las funciones pasarán a modo Auto (Automático) de forma predeterminada. Puede aplicar esta restricción en toda su organización mediante [políticas de control de servicios \(SCP\)](#). Si debe revertir una función a una versión anterior en tiempo de ejecución, puede admitir una excepción de política caso por caso.

# Recuperar datos sobre las funciones de Lambda que utilizan un tiempo de ejecución obsoleto

Cuando un tiempo de ejecución de Lambda está a punto de quedar obsoleto, Lambda lo informa por correo electrónico y envía notificaciones en AWS Health Dashboard y Trusted Advisor. En estos correos electrónicos y notificaciones se enumeran las versiones \$LATEST de las funciones que utilizan el tiempo de ejecución. Para enumerar todas las versiones de las funciones que utilizan un tiempo de ejecución determinado, puede usar la AWS Command Line Interface (AWS CLI) o uno de los AWS SDK.

Si tiene una gran cantidad de funciones que utilizan un tiempo de ejecución obsoleto, también puede utilizar la AWS CLI o los AWS SDK como ayuda para priorizar las actualizaciones de las funciones que más se invocan.

Consulte las siguientes secciones para aprender a usar la AWS CLI y los AWS SDK para recopilar datos sobre las funciones que utilizan un tiempo de ejecución determinado.

## Lista de las versiones de funciones que utilizan un tiempo de ejecución determinado

Para usar la AWS CLI para obtener una lista de todas las versiones de las funciones que utilizan un tiempo de ejecución determinado, ejecute el siguiente comando: Reemplace `RUNTIME_IDENTIFIER` por el nombre del tiempo de ejecución que está por caducar y elija su propia Región de AWS. Para mostrar solo las versiones de la función \$LATEST, omita `--function-version ALL` en el comando.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

### Tip

El comando de ejemplo muestra las funciones de la región `us-east-1` para un determinado Cuenta de AWS. Deberá repetir este comando para cada región en la que su cuenta tenga funciones y para cada uno de sus Cuentas de AWS.

También puede obtener una lista de las funciones que utilizan un tiempo de ejecución determinado mediante uno de los AWS SDK. El siguiente código de ejemplo utiliza la V3 del AWS SDK for

JavaScript y el AWS SDK for Python (Boto3) para devolver una lista de los ARN de funciones de las funciones que utilizan un tiempo de ejecución determinado. El código de ejemplo también devuelve el grupo de registro de CloudWatch para cada una de las funciones de la lista. Puede usar este grupo de registro para buscar la fecha de la última invocación de la función. Consulte la siguiente sección [the section called “Identificación de las funciones invocadas más recientemente y con mayor frecuencia”](#) para obtener más información.

## Node.js

Example Código JavaScript para obtener una lista de las funciones que utilizan un tiempo de ejecución determinado

```
import { LambdaClient, ListFunctionsCommand } from "@aws-sdk/client-lambda";
const lambdaClient = new LambdaClient();

const command = new ListFunctionsCommand({
  FunctionVersion: "ALL",
  MaxItems: 50
});
const response = await lambdaClient.send(command);

for (const f of response.Functions){
  if (f.Runtime == '<your_runtime>'){ // Use the runtime id, e.g. 'nodejs18.x' or
  'python3.9'
    console.log(f.FunctionArn);
    // get the CloudWatch log group of the function to
    // use later for finding the last invocation date
    console.log(f.LoggingConfig.LogGroup);
  }
}
// If your account has more functions than the specified
// MaxItems, use the returned pagination token in the
// next request with the 'Marker' parameter
if ('NextMarker' in response){
  let paginationToken = response.NextMarker;
}
```

## Python

Example Código Python para obtener una lista de las funciones que utilizan un tiempo de ejecución determinado

```
import boto3

lambda_client = boto3.client('lambda')

response = lambda_client.list_functions(
    FunctionVersion='ALL',
    MaxItems=50
)

for function in response['Functions']:
    if function['Runtime'] == '<your_runtime>': # Use the runtime id, e.g.
        'nodejs18.x' or 'python3.9'
        print(function['FunctionArn'])
        # Print the CloudWatch log group of the function
        # to use later for finding last invocation date
        print(function['LoggingConfig']['LogGroup'])

# If your account has more functions than the specified
# MaxItems, use the returned pagination token in the
# next request with the 'Marker' parameter
if 'NextMarker' in response:
    pagination_token = response['NextMarker']
```

Para obtener más información sobre el uso de un AWS SDK para enumerar las funciones mediante la acción [ListFunctions](#), consulte la [documentación del SDK](#) del lenguaje de programación que prefiera.

También puede utilizar la característica Consultas avanzadas del AWS Config para enumerar todas las funciones que utilizan un tiempo de ejecución afectado. Esta consulta solo devuelve las versiones \$LATEST de la función, pero puede agregar consultas para enumerar funciones en todas las regiones y en varias Cuentas de AWS con un solo comando. Para obtener más información, lea [Consulta sobre el estado de la configuración actual de los recursos del AWS Auto Scaling](#) en la Guía del AWS Config para desarrolladores.



## Identificación de las funciones invocadas más recientemente y con mayor frecuencia

Si su Cuenta de AWS incluye funciones que utilizan un tiempo de ejecución que está por quedar obsoleto, quizá desee priorizar la actualización de las funciones que se invocan con frecuencia o que se invocaron recientemente.

Si solo dispone de unas pocas funciones, puede utilizar la consola de los registros de CloudWatch para recopilar esta información consultando los flujos de registro de sus funciones. Para obtener más información, consulte [Ver datos de registro enviados a los registros de CloudWatch](#).

Para ver la cantidad de invocaciones de funciones recientes, también puede utilizar la información de métricas de CloudWatch que se muestra en la consola de Lambda. Para ver esta información, siga estas instrucciones:

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función de la que desee consultar las estadísticas de invocación.
3. Elija la pestaña Monitor (Monitorear).
4. Establezca el periodo del que desea ver las estadísticas utilizando el selector de rango de fechas. Las invocaciones recientes se muestran en el panel Invocaciones.

Para las cuentas con una mayor cantidad de funciones, puede ser más eficiente recopilar estos datos mediante programación con la AWS CLI o uno de los AWS SDK mediante las acciones de la API [DescribeLogStreams](#) y [GetMetricStatistics](#).

En los siguientes ejemplos, se proporcionan fragmentos de código que utilizan la V3 del AWS SDK for JavaScript y el AWS SDK for Python (Boto3) para identificar la fecha de la última invocación de una función determinada y establecer la cantidad de invocaciones de una función determinada en los últimos 14 días.

Node.js

Example Código JavaScript para encontrar la hora de la última invocación de una función

```
import { CloudWatchLogsClient, DescribeLogStreamsCommand } from "@aws-sdk/client-cloudwatch-logs";
const cloudWatchLogsClient = new CloudWatchLogsClient();
const command = new DescribeLogStreamsCommand({
```

```

    logGroupName: '<your_log_group_name>',
    orderBy: 'LastEventTime',
    limit: 1
  });
  try {
    const response = await cloudWatchLogsClient.send(command);
    const lastEventTimestamp = response.logStreams.length > 0 ?
      response.logStreams[0].lastEventTimestamp : null;
    // Convert the UNIX timestamp to a human-readable format for display
    const date = new Date(lastEventTimestamp).toLocaleDateString();
    const time = new Date(lastEventTimestamp).toLocaleTimeString();
    console.log(`${date} ${time}`);
  } catch (e){
    console.error('Log group not found.')
  }
}

```

## Python

Example Código Python para encontrar la hora de la última invocación de una función

```

import boto3
from datetime import datetime

cloudwatch_logs_client = boto3.client('logs')

response = cloudwatch_logs_client.describe_log_streams(
    logGroupName='<your_log_group_name>',
    orderBy='LastEventTime',
    limit=1
)

try:
    if len(response['logStreams']) > 0:
        last_event_timestamp = response['logStreams'][0]['lastEventTimestamp']
        print(datetime.fromtimestamp(last_event_timestamp/1000)) # Convert timestamp
        from ms to seconds
    else:
        last_event_timestamp = None
except:
    print('Log group not found')

```

**i** Tip

Puede encontrar el nombre del grupo de registro de la función mediante la operación de la API [ListFunctions](#). Para ver ejemplos de cómo hacerlo, consulte el código en [the section called “Lista de las versiones de funciones que utilizan un tiempo de ejecución determinado”](#).

## Node.js

Example Código JavaScript para encontrar la cantidad de invocaciones en los últimos 14 días

```
import { CloudWatchClient, GetMetricStatisticsCommand } from "@aws-sdk/client-cloudwatch";
const cloudWatchClient = new CloudWatchClient();
const command = new GetMetricStatisticsCommand({
  Namespace: 'AWS/Lambda',
  MetricName: 'Invocations',
  StartTime: new Date(Date.now()-86400*1000*14), // 14 days ago
  EndTime: new Date(Date.now()),
  Period: 86400 * 14, // 14 days.
  Statistics: ['Sum'],
  Dimensions: [{
    Name: 'FunctionName',
    Value: '<your_function_name>'
  }]
});
const response = await cloudWatchClient.send(command);
const invokesInLast14Days = response.Datapoints.length > 0 ?
  response.Datapoints[0].Sum : 0;

console.log('Number of invocations: ' + invokesInLast14Days);
```

## Python

Example Código Python para encontrar la cantidad de invocaciones en los últimos 14 días

```
import boto3
from datetime import datetime, timedelta

cloudwatch_client = boto3.client('cloudwatch')

response = cloudwatch_client.get_metric_statistics(
```

```
Namespace='AWS/Lambda',
MetricName='Invocations',
Dimensions=[
    {
        'Name': 'FunctionName',
        'Value': '<your_function_name>'
    },
],
StartTime=datetime.now() - timedelta(days=14),
EndTime=datetime.now(),
Period=86400 * 14, # 14 days
Statistics=[
    'Sum'
]
)

if len(response['Datapoints']) > 0:
    invokes_in_last_14_days = int(response['Datapoints'][0]['Sum'])
else:
    invokes_in_last_14_days = 0

print(f'Number of invocations: {invokes_in_last_14_days}')
```

## Modificación del entorno de tiempo de ejecución

Puede utilizar [extensiones internas](#) para modificar el proceso de tiempo de ejecución. Las extensiones internas no son procesos independientes, se ejecutan como parte del proceso de tiempo de ejecución.

Lambda proporciona específica [variables de entorno](#) específicas del lenguaje que puede configurar para agregar opciones y herramientas al tiempo de ejecución. Lambda también proporciona [scripts de encapsulador](#), que permiten a Lambda delegar el inicio del tiempo de ejecución a su script. Puede crear un script de encapsulador para personalizar el comportamiento de inicio del tiempo de ejecución.

## Variables de entorno específicas del lenguaje

Lambda admite formas de solo configuración para habilitar la precarga del código durante la inicialización de la función a través de las siguientes variables de entorno específicas del lenguaje:

- `JAVA_TOOL_OPTIONS`: en Java, Lambda admite esta variable de entorno para establecer variables de línea de comandos adicionales en Lambda. Esta variable de entorno le permite especificar la inicialización de herramientas, específicamente el lanzamiento de agentes de lenguaje de programación nativo o Java utilizando las opciones `agentlib` o `javaagent`. Para obtener más información, consulte la [variable de entorno de JAVA\\_TOOL\\_OPTIONS](#).
- `NODE_OPTIONS`: disponible en los [tiempos de ejecución de Node.js](#).
- `DOTNET_STARTUP_HOOKS`: en .NET Core 3.1 y superior, esta variable de entorno especifica una ruta a un ensamblado (dll) que Lambda puede usar.

El uso de variables de entorno específicas del lenguaje es la forma preferida de establecer propiedades de inicio.

## Scripts de encapsulador

Puede crear un script del encapsulador para personalizar el comportamiento de inicio en tiempo de ejecución de su función de Lambda. Un script de encapsulador le permite establecer parámetros de configuración que no se pueden establecer mediante variables de entorno específicas del lenguaje.

**Note**

Las invocaciones pueden producir un error si el script de encapsulador no comienza correctamente el proceso de tiempo de ejecución.

Los scripts de encapsulador son compatibles con todos los [tiempos de ejecución de Lambda](#) nativos. Los scripts de encapsulador no son compatibles con [Tiempos de ejecución exclusivos del sistema operativo](#) (la familia de tiempos de ejecución de provided).

Cuando utiliza un script de encapsulador para su función, Lambda inicia el tiempo de ejecución utilizando el script. Lambda envía al script la ruta al intérprete y todos los argumentos originales para el inicio de tiempo de ejecución estándar. El script puede ampliar o transformar el comportamiento de inicio del programa. Por ejemplo, el script puede inyectar y modificar argumentos, establecer variables de entorno o capturar métricas, errores y otra información de diagnóstico.

Para especificar el script mediante el establecimiento del valor de la variable de entorno `AWS_LAMBDA_EXEC_WRAPPER` como la ruta del sistema de archivos de un binario o script ejecutable.

### Ejemplo: crear y usar un script de encapsulador con Python 3.8

En el ejemplo siguiente, se crea un script de encapsulador para comenzar el intérprete de Python con la opción `-X importtime`. Cuando ejecuta la función, Lambda genera una entrada de registro para mostrar la duración del tiempo de importación para cada importación.

Para crear y usar un script de encapsulador con Python 3.8

1. Para crear el script de encapsulador, pegue el código siguiente en un archivo denominado `importtime_wrapper`:

```
#!/bin/bash

# the path to the interpreter and all of the originally intended arguments
args=("$@")

# the extra options to pass to the interpreter
extra_args="-X" "importtime"

# insert the extra options
args=("${args[@]:0:$#-1}" "${extra_args[@]}" "${args[@]: -1}")
```

```
# start the runtime with the extra options
exec "${args[@]}"
```

2. Para conceder permisos ejecutables al script, escriba `chmod +x importtime_wrapper` desde la línea de comandos.
3. Implemente el script como una [capa de Lambda](#).
4. Luego, se crea una función con la consola de Lambda.
  - a. Abra la [consola de Lambda](#).
  - b. Elija Crear función.
  - c. En Basic information (Información básica), para Function name (Nombre de función), escriba **wrapper-test-function**.
  - d. En Tiempo de ejecución, seleccione Python 3.8.
  - e. Seleccione Crear función.
5. Agregue la capa a la función.
  - a. Elija su función y, a continuación, elija Código si aún no está seleccionada.
  - b. Elija Add a layer (Añadir una capa).
  - c. En Choose a layer (Elegir una capa), elija el Name (Nombre) y la Version (Versión) de la capa compatible que ha creado antes.
  - d. Elija Añadir.
6. Agregue el código y la variable de entorno a la función.
  - a. En el editor de código de función, pegue el siguiente código de función:

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

- b. Seleccione Guardar.

- c. En Variables de entorno, elija Editar.
  - d. Elija Add environment variable (Añadir variable de entorno).
  - e. En Clave, escriba `AWS_LAMBDA_EXEC_WRAPPER`.
  - f. En Valor, introduzca `/opt/importtime_wrapper`.
  - g. Seleccione Guardar.
7. Para ejecutar la función, elija Test (Prueba).

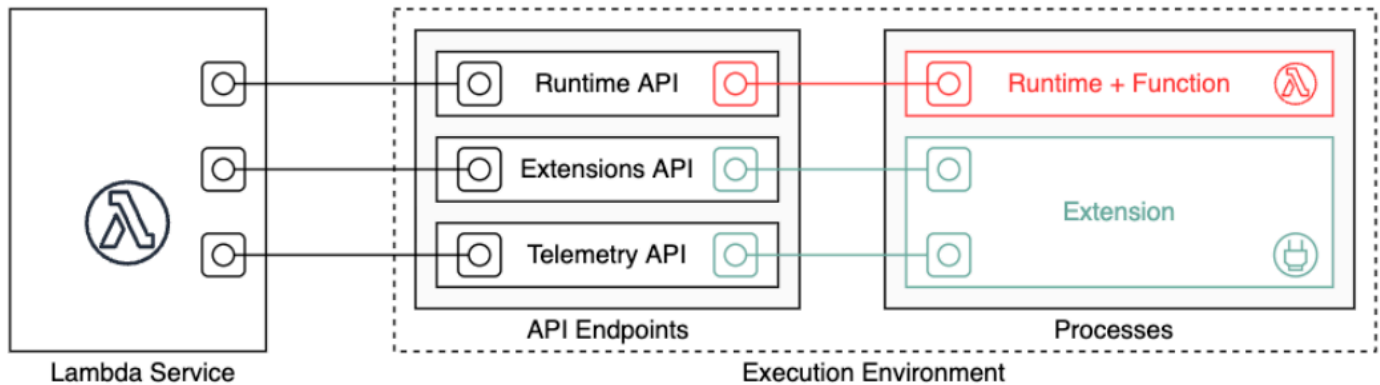
Debido a que el script de encapsulador comenzó el intérprete de Python con la opción `-X importtime`, los registros muestran el tiempo necesario para cada importación. Por ejemplo:

```
...
2020-06-30T18:48:46.780+01:00 import time: 213 | 213 | simplejson
2020-06-30T18:48:46.780+01:00 import time: 50 | 263 | simplejson.raw_json
...
```



# Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados

AWS Lambda proporciona una API HTTP para que los [tiempos de ejecución personalizados](#) puedan recibir eventos de invocación desde Lambda y enviar los datos de respuesta dentro del [entorno de ejecución](#) de Lambda. Esta sección contiene la referencia de la API para la API de tiempo de ejecución de Lambda.



La especificación de OpenAPI para la versión de la API de tiempo de ejecución 2018-06-01 está disponible en: [runtime-api.zip](#).

Para crear una URL de solicitud de API, los tiempos de ejecución obtienen el punto de enlace de la API de la variable de entorno `AWS_LAMBDA_RUNTIME_API`; agregue la versión de la API y la ruta de recurso deseada.

## Example Solicitud

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

## Métodos de API

- [Siguiete invocación](#)
- [Respuesta de la invocación](#)
- [Error de inicialización](#)
- [Error de invocación](#)

## Siguiente invocación

Ruta – `/runtime/invocation/next`

Método: GET

El tiempo de ejecución envía este mensaje a Lambda para solicitar un evento de invocación. El cuerpo de la respuesta contiene la carga a partir de la invocación, la cual es un documento JSON que contiene datos de eventos del desencadenador de la función. Los encabezados de la respuesta contienen datos adicionales sobre la invocación.

### Encabezados de respuesta

- `Lambda-Runtime-Aws-Request-Id`: el ID de solicitud, el cual identifica la solicitud que ha desencadenado la invocación de la función.

Por ejemplo, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.

- `Lambda-Runtime-Deadline-Ms`: la fecha en la que la función agota su tiempo de espera en milisegundos de tiempo Unix.

Por ejemplo, `1542409706888`.

- `Lambda-Runtime-Invoked-Function-Arn`: el ARN de la función de Lambda, la versión o el alias especificado en la invocación.

Por ejemplo, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.

- `Lambda-Runtime-Trace-Id`: el [encabezado de rastreo AWS X-Ray](#).

Por ejemplo, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.

- `Lambda-Runtime-Client-Context`: en invocaciones desde AWS Mobile SDK, datos sobre la aplicación cliente y el dispositivo.
- `Lambda-Runtime-Cognito-Identity`: para invocaciones desde AWS Mobile SDK, datos sobre el proveedor de identidades de Amazon Cognito.

No establezca un tiempo de espera en la solicitud GET, ya que la respuesta puede retrasarse. Mientras Lambda arranca el tiempo de ejecución y mientras el tiempo de ejecución tiene un evento para devolver, el proceso de tiempo de ejecución se congela durante varios segundos.

El ID de solicitud hace un seguimiento de la invocación dentro de Lambda. Utilícelo para especificar la invocación al enviar la respuesta.

El encabezado de rastreo contiene el ID de rastreo, el ID principal y la decisión de muestreo. Si se muestrea la solicitud, lo hace Lambda o un servicio ascendente. El tiempo de ejecución deberá definir el valor `_X_AMZN_TRACE_ID` con el valor del encabezado. El X-Ray SDK lee esto para obtener los ID y determinar si hay que rastrear la solicitud.

## Respuesta de la invocación

Ruta – `/runtime/invocation/AwsRequestId/response`

Método: POST

Después de que la función se ha ejecutado hasta su finalización, el tiempo de ejecución envía una respuesta de invocación a Lambda. En el caso de invocaciones síncronas, Lambda envía la respuesta al cliente.

Example Solicitud correcta

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/${REQUEST_ID}/response" -d "SUCCESS"
```

## Error de inicialización

Si la función devuelve un error o el tiempo de ejecución encuentra un error durante la inicialización, el tiempo de ejecución utiliza este método para informar del error a Lambda.

Ruta – `/runtime/init/error`

Método: POST

Encabezados

`Lambda-Runtime-Function-Error-Type`: tipo de error que encontró el tiempo de ejecución.

Obligatorio: no

Este encabezado consta de un valor de cadena. Lambda acepta cualquier cadena, pero recomendamos un formato de `<category.reason>`. Por ejemplo:

- `Runtime.NoSuchHandler`
- `Runtime.APIKeyNotFound`
- `Runtime.ConfigInvalid`
- `Runtime.UnknownReason`

Body parameters (Parámetros del cuerpo)

`ErrorRequest`: información sobre el error. Obligatorio: no

Este campo es un objeto JSON con la siguiente estructura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
  stackTrace: array of strings
}
```

Tenga en cuenta que Lambda acepta cualquier valor para `errorType`.

En el ejemplo siguiente, se muestra un mensaje de error de función de Lambda en el que la función no pudo analizar los datos de evento proporcionados en la invocación.

Example Error de la función

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException",
  "stackTrace": [ ]
}
```

Parámetros del cuerpo de la respuesta

- `StatusResponse` – Cadena. Información de estado, enviada con códigos de respuesta 202.
- `ErrorResponse`: información adicional sobre el error, enviada con los códigos de respuesta de error. `ErrorResponse` contiene un tipo de error y un mensaje de error.

Códigos de respuesta

- 202: aceptada

- 403: prohibido
- 500: error en contenedor. Estado no recuperable. El tiempo de ejecución debe salir rápidamente.

### Example Solicitud de error de inicialización

```
ERROR="{\"errorMessage\" : \"Failed to load function.\", \"errorType\" :  
  \"InvalidFunctionException\"}"  
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR" --  
header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

## Error de invocación

Si la función devuelve un error o el tiempo de ejecución encuentra un error, el tiempo de ejecución utiliza este método para informar del error a Lambda.

Ruta – `/runtime/invocation/AwsRequestId/error`

Método: POST

Encabezados

`Lambda-Runtime-Function-Error-Type`: tipo de error que encontró el tiempo de ejecución.

Obligatorio: no

Este encabezado consta de un valor de cadena. Lambda acepta cualquier cadena, pero recomendamos un formato de `<category.reason>`. Por ejemplo:

- `Runtime.NoSuchHandler`
- `Runtime.APIKeyNotFound`
- `Runtime.ConfigInvalid`
- `Runtime.UnknownReason`

Body parameters (Parámetros del cuerpo)

`ErrorRequest`: información sobre el error. Obligatorio: no

Este campo es un objeto JSON con la siguiente estructura:

```
{
```

```

    errorMessage: string (text description of the error),
    errorType: string,
    stackTrace: array of strings
}

```

Tenga en cuenta que Lambda acepta cualquier valor para `errorType`.

En el ejemplo siguiente, se muestra un mensaje de error de función de Lambda en el que la función no pudo analizar los datos de evento proporcionados en la invocación.

### Example Error de la función

```

{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException",
  "stackTrace": [ ]
}

```

### Parámetros del cuerpo de la respuesta

- `StatusResponse` – Cadena. Información de estado, enviada con códigos de respuesta 202.
- `ErrorResponse`: información adicional sobre el error, enviada con los códigos de respuesta de error. `ErrorResponse` contiene un tipo de error y un mensaje de error.

### Códigos de respuesta

- 202: aceptada
- 400: solicitud errónea
- 403: prohibido
- 500: error en contenedor. Estado no recuperable. El tiempo de ejecución debe salir rápidamente.

### Example Solicitud errónea

```

REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
ERROR="{\"errorMessage\" : \"Error parsing event data.\", \"errorType\" :
  \"InvalidEventDataException\"}"
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/error"
-d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"

```

# Cuándo utilizar los tiempos de ejecución exclusivos del sistema operativo de Lambda

Lambda proporciona [tiempos de ejecución gestionados](#) para Java, Python, Node.js, .NET y Ruby. Para crear funciones de Lambda en un lenguaje de programación que no esté disponible como tiempo de ejecución gestionado, utilice un tiempo de ejecución exclusivo del sistema operativo (la familia de tiempos de ejecución provided). Existen tres casos de uso principales para los tiempos de ejecución exclusivos del sistema operativo:

- **Compilación nativa anticipada (AOT):** lenguajes como Go, Rust y C++ se compilan de forma nativa en un binario ejecutable, que no requiere un tiempo de ejecución de lenguaje específico. Estos lenguajes solo necesitan un entorno de sistema operativo en el que se pueda ejecutar el binario compilado. También puede usar tiempos de ejecución exclusivos del sistema operativo de Lambda para implementar binarios compilados con .NET Native AOT y Java GraalVM Native.

Debe incluir un cliente de interfaz de tiempo de ejecución en el binario. El cliente de la interfaz del tiempo de ejecución llama a [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#) para recuperar las invocaciones de funciones y, a continuación, llama al controlador de funciones. Lambda proporciona clientes de interfaz de tiempo de ejecución para [Go](#), [.NET Native AOT](#), [C++](#) y [Rust](#) (experimental).

Debe compilar el binario para un entorno Linux y para la misma arquitectura de conjunto de instrucciones que planea usar para la función (x86\_64 o arm64).

- **Tiempos de ejecución de terceros:** puede ejecutar funciones de Lambda con tiempos de ejecución estándar, como [Bref](#) para PHP o [Swift AWS Lambda Runtime](#) para Swift.
- **Tiempos de ejecución personalizados:** puede crear su propio tiempo de ejecución para un lenguaje (o una versión de un lenguaje) para el que Lambda no proporcione un tiempo de ejecución gestionado, como Node.js 19. Para obtener más información, consulte [Creación de un tiempo de ejecución personalizado para AWS Lambda](#). Este es el caso de uso menos común para los tiempos de ejecución exclusivos del sistema operativo.

Lambda admite los siguientes tiempos de ejecución exclusivos del sistema operativo de Ruby.

## Exclusivo del sistema operativo

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Tiempo de ejecución exclusivo del sistema operativo	provided.a12023	Amazon Linux 2023			
Tiempo de ejecución exclusivo del sistema operativo	provided.a12	Amazon Linux 2			

El tiempo de ejecución de Amazon Linux 2023 (`provided.a12023`) ofrece varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como `glibc`.

El tiempo de ejecución `provided.a12023` utiliza `dnf` como administrador de paquetes en lugar de `yum`, que es el administrador de paquetes predeterminado en Amazon Linux 2. Para obtener más información sobre las diferencias entre `provided.a12023` y `provided.a12`, consulte [Presentación del tiempo de ejecución de Amazon Linux 2023 AWS Lambda](#) en el Blog de informática de AWS.

## Creación de un tiempo de ejecución personalizado para AWS Lambda

Puede implementar tiempos de ejecución de AWS Lambda con cualquier lenguaje de programación. El tiempo de ejecución es un programa que ejecuta un método de controlador de función de Lambda cuando se invoca la función. El tiempo de ejecución se puede incluir en el paquete de implementación de la función o se puede distribuir en una [capa](#). Cuando cree la función de Lambda, elija un [tiempo de ejecución exclusivo del sistema operativo](#) (la familia de tiempos de ejecución `provided`).



### Note

La creación de un tiempo de ejecución personalizado es un caso de uso avanzado. Si busca información sobre la compilación en un binario nativo o el uso de un tiempo de ejecución estándar de terceros, consulte [Cuándo utilizar los tiempos de ejecución exclusivos del sistema operativo de Lambda](#).

Para ver un recorrido por el proceso de implementación de un tiempo de ejecución personalizado, consulte [Tutorial: cómo crear un tiempo de ejecución personalizado](#). También puede estudiar un tiempo de ejecución personalizado implementado en C++ en [aws-lambda-cpp](#) en GitHub.

### Temas

- [Requisitos](#)
- [Implementación de la transmisión de respuestas en un entorno de ejecución personalizado](#)

### Requisitos

Los tiempos de ejecución personalizados deben completar determinadas tareas de inicialización y procesamiento. El tiempo de ejecución ejecuta el código de configuración de la función, lee el nombre del controlador a partir de una variable de entorno y lee los eventos de invocación desde la API de tiempo de ejecución de Lambda. El tiempo de ejecución transfiere los datos de los eventos al controlador de la función y publica la respuesta desde el controlador de vuelta a Lambda.

### Tareas de inicialización

Las tareas de inicialización se ejecutan una vez [por instancia de la función](#) para preparar el entorno para gestionar invocaciones.

- Retrieve settings (Recuperar opciones de configuración): leer variables de entorno para obtener detalles acerca de la función y el entorno.
  - `_HANDLER`: la ubicación del controlador, a partir de la configuración de la función. El formato estándar es `file.method`, donde `file` es el nombre del archivo sin extensión, y `method` es el nombre de un método o una función que se define en el archivo.
  - `LAMBDA_TASK_ROOT`: el directorio que contiene el código de la función.
  - `AWS_LAMBDA_RUNTIME_API`: el host y el puerto de la API de tiempo de ejecución.

Para obtener una lista completa de variables disponibles, consulte [Variables definidas de entorno de tiempo de ejecución](#).

- Inicializar la función: cargar el archivo de controlador y ejecutar cualquier código global o estático en él. Las funciones deben crear los recursos estáticos, como clientes SDK y conexiones de bases de datos, una vez, y después volver a utilizarlos para varias invocaciones.
- Administración de errores: si se produce un error, llamar a la API de [error de inicialización](#) y salir de forma inmediata.

La inicialización se tiene en cuenta al calcular el tiempo de ejecución y el tiempo de espera que se factura. Cuando una ejecución activa la inicialización de una nueva instancia de la función, puede ver el tiempo de inicialización en los registros y el [rastreo de AWS X-Ray](#).

### Example registro

```
REPORT RequestId: f8ac1208... Init Duration: 48.26 ms   Duration: 237.17 ms   Billed
Duration: 300 ms   Memory Size: 128 MB   Max Memory Used: 26 MB
```

### Procesamiento de tareas

Mientras se ejecuta, el tiempo de ejecución utiliza la [interfaz de tiempo de ejecución de Lambda](#) para administrar eventos entrantes e informar sobre errores. Después de completar las tareas de inicialización, el tiempo de ejecución procesa los eventos entrantes en bucle. En el código de tiempo de ejecución, realice los siguientes pasos en orden.

- Get an event (Obtener un evento): llamar a la API de la [siguiente invocación](#) para obtener el siguiente evento. El cuerpo de la respuesta contiene los datos del evento. Los encabezados de respuesta contienen el ID de la solicitud y otra información.
- Propagate the tracing header (Propagar el encabezado de rastreo): obtenga el encabezado de rastreo de X-Ray desde el encabezado `Lambda-Runtime-Trace-Id` en la respuesta de la API. Establezca localmente la variable de entorno `_X_AMZN_TRACE_ID` con el mismo valor. El SDK de X-Ray utiliza este valor para conectar datos de rastreo entre servicios.
- Create a context object (Crear un objeto context): cree un objeto con información de contexto a partir de las variables de entorno y los encabezados en la respuesta de la API.
- Invoke the function handler (Invocar el controlador de la función): transfiera el evento y el objeto context al controlador.

- Handle the response (Administrar la respuesta): llame a la API de [respuesta de invocación](#) para publicar la respuesta desde el controlador.
- Handle errors (Administrar errores): si se produce un error, llamar a la API de [error de invocación](#).
- Cleanup (Limpiar): libere recursos no utilizados, envíe datos a otros servicios o realice tareas adicionales antes de obtener el siguiente evento.

## Punto de entrada

El punto de entrada de un tiempo de ejecución personalizado es un archivo ejecutable llamado `bootstrap`. El archivo de arranque puede ser el tiempo de ejecución o puede invocar otro archivo que cree el tiempo de ejecución. Si la raíz del paquete de implementación no contiene un archivo con el nombre `bootstrap`, Lambda busca el archivo en las capas de la función. Si el archivo `bootstrap` no existe o no es ejecutable, la función devuelve el error `Runtime.InvalidEntryPoint` tras la invocación.

Este es un ejemplo de un archivo `bootstrap` que utiliza una versión empaquetada de Node.js para ejecutar un tiempo de ejecución de JavaScript en un archivo aparte llamado `runtime.js`.

## Example bootstrap

```
#!/bin/sh
cd $LAMBDA_TASK_ROOT
./node-v11.1.0-linux-x64/bin/node runtime.js
```

## Implementación de la transmisión de respuestas en un entorno de ejecución personalizado

Para las [funciones de transmisión de respuestas](#), los puntos de conexión de `response` y `error` han modificado ligeramente su comportamiento, lo que permite que el tiempo de ejecución transmita respuestas parciales al cliente y devuelva las cargas en fragmentos. Para obtener más información sobre el comportamiento específico, consulte lo siguiente:

- `/runtime/invocation/AwsRequestId/response`: propaga el encabezado `Content-Type` desde el tiempo de ejecución para enviarlo al cliente. Lambda devuelve la carga de respuesta en fragmentos mediante la codificación de transferencia fragmentada de HTTP/1.1. El tamaño máximo de transmisión de respuesta es de 20 MiB. Para transmitir la respuesta a Lambda, el tiempo de ejecución debe realizar lo siguiente:

- Establecer el encabezado `HTTP Lambda-Runtime-Function-Response-Mode` en `streaming`.
- Establezca el encabezado `Transfer-Encoding` en `chunked`.
- Escribir la respuesta de acuerdo con la especificación de codificación de transferencia fragmentada de HTTP/1.1.
- Cerrar la conexión subyacente después de que la respuesta se haya escrito correctamente.
- `/runtime/invocation/AwsRequestId/error`: el tiempo de ejecución puede utilizar este punto de conexión para informar sobre errores de función o tiempo de ejecución a Lambda, que también acepta el encabezado `Transfer-Encoding`. Solo se puede llamar a este punto de conexión antes de que el tiempo de ejecución comience a enviar una respuesta de invocación.
- Informe los errores intermedios mediante los tráileres de errores en `/runtime/invocation/AwsRequestId/response`: el tiempo de ejecución puede adjuntar opcionalmente los encabezados finales de HTTP con los nombres `Lambda-Runtime-Function-Error-Type` y `Lambda-Runtime-Function-Error-Body` para informar sobre los errores que se producen después de haber empezado a escribir la respuesta de invocación. Lambda trata esto como una respuesta correcta y reenvía los metadatos de error proporcionados por el tiempo de ejecución al cliente.

#### Note

Para adjuntar encabezados finales, el tiempo de ejecución debe establecer el valor del encabezado `Trailer` al principio de la solicitud HTTP. Este es un requisito de la especificación de codificación de transferencia fragmentada de HTTP/1.1.

- `Lambda-Runtime-Function-Error-Type`: el tipo de error que encontró el tiempo de ejecución. Este encabezado consta de un valor de cadena. Lambda acepta cualquier cadena, pero recomendamos un formato de `<category.reason>`. Por ejemplo, `Runtime.APIKeyNotFound`.
- `Lambda-Runtime-Function-Error-Body`: información sobre el error codificada en Base64.

## Tutorial: cómo crear un tiempo de ejecución personalizado

En este tutorial creará una función de Lambda con un tiempo de ejecución personalizado. Para empezar incluirá el tiempo de ejecución en el paquete de implementación de la función. A

continuación, lo migrará a una capa que gestiona independientemente de la función. Por último, compartirá la capa de tiempo de ejecución globalmente actualizando su política de permisos basados en recursos.

## Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#), para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, `zip`) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.

Necesita un rol de IAM para crear una función de Lambda. El rol necesita permiso para enviar registros a los Registros de CloudWatch y acceder al servicio de AWS que utiliza su función. Si no tiene un rol de para el desarrollo de funciones, cree uno.

Para crear un rol de ejecución

1. Abra la [página Roles](#) en la consola de IAM.

2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Trusted entity (Entidad de confianza).-Lambda:.
  - Permisos: AWSLambdaBasicExecutionRole.
  - Nombre de rol: **lambda-role**.

La política AWSLambdaBasicExecutionRole tiene permisos que la función necesita para escribir registros a Registros de CloudWatch.

## Crear una función

Cree una función de Lambda con un tiempo de ejecución personalizado. En este ejemplo se incluyen dos archivos: un archivo `bootstrap` de tiempo de ejecución y un controlador de la función. Ambos se implementan en Bash.

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Cree un nuevo archivo denominado `bootstrap`. Este es el tiempo de ejecución personalizado.

### Example bootstrap

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event. The HTTP request will block until one is received
  EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")
```

```
# Extract request ID by scraping response headers received above
REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

# Run the handler function from the script
RESPONSE=$(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

# Send the response
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "$RESPONSE"
done
```

El tiempo de ejecución carga un script de función desde el paquete de implementación. Utiliza dos variables para localizar el script. `LAMBDA_TASK_ROOT` le indica dónde se extrajo el paquete y `_HANDLER` incluye el nombre del script.

Luego de que el tiempo de ejecución carga el script de la función, utiliza la API de tiempo de ejecución para recuperar un evento de invocación de Lambda, pasa el evento al controlador y publica la respuesta de vuelta en Lambda. Para obtener el ID de la solicitud, el tiempo de ejecución guarda los encabezados de la respuesta de la API en un archivo temporal y lee el encabezado `Lambda-Runtime-Aws-Request-Id` del archivo.

#### Note

Los tiempos de ejecución se encargan de algunas cosas más, como gestionar errores y proporcionar información de contexto al controlador. Para obtener más información, consulte [Requisitos](#).

3. Cree un script para la función. El script del ejemplo a continuación, define una función de controlador que toma datos de eventos, los registra en `stderr` y los devuelve.

#### Example function.sh

```
function handler () {
  EVENT_DATA=$1
  echo "$EVENT_DATA" 1>&2;
  RESPONSE="Echoing request: '$EVENT_DATA'"

  echo $RESPONSE
}
```

El directorio `runtime-tutorial` debe tener ahora el siguiente aspecto:

```
runtime-tutorial
# bootstrap
# function.sh
```

- Haga los archivos ejecutables y añádalos a un archivo `.zip`. Este es el paquete de implementación.

```
chmod 755 function.sh bootstrap
zip function.zip function.sh bootstrap
```

- Cree una función llamada `bash-runtime`. Para `--role`, introduzca el ARN de su [rol de ejecución](#) de Lambda.

```
aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime
provided.al2023 \
--role arn:aws:iam::123456789012:role/lambda-role
```

- invoque la función.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
response.txt --cli-binary-format raw-in-base64-out
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver una respuesta como la siguiente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

- Verifique la respuesta.



```
cat response.txt
```

Debería ver una respuesta como la siguiente:

```
Echoing request: '{"text":"Hello"}'
```

## Crear una capa

Para separar el código de tiempo de ejecución del código de la función, cree una capa que solo contenga el tiempo de ejecución. Las capas le permiten desarrollar las dependencias de la función de forma independiente y, si utiliza la misma capa con varias funciones, podrá hacer un menor uso del almacenamiento. Para obtener más información, consulte [Administración de las dependencias de Lambda con capas](#).

1. Cree un archivo `.zip` que contenga el archivo `bootstrap`.

```
zip runtime.zip bootstrap
```

2. Cree una capa con el comando [publish-layer-version](#).

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://  
runtime.zip
```

Esto crea la primera versión de la capa.

## Actualización de la función

Para utilizar la capa de tiempo de ejecución con la función, configure la función para que use la capa, y elimine de ella el código de tiempo de ejecución.

1. Actualice la configuración de la función para incluir la capa.

```
aws lambda update-function-configuration --function-name bash-runtime \  
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
```

Esto agrega el tiempo de ejecución a la función en el directorio /opt. Para garantizar que Lambda utilice el tiempo de ejecución de la capa, debe eliminar el bootstrap del paquete de implementación de la función, como se muestra en los dos pasos a continuación.

2. Cree un archivo .zip que contenga el código de la función.

```
zip function-only.zip function.sh
```

3. Actualice el código de la función para que incluya solo el script del controlador.

```
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://function-only.zip
```

4. Invoque la función para confirmar que funciona con la capa de tiempo de ejecución.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}' response.txt --cli-binary-format raw-in-base64-out
```

La opción cli-binary-format es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver una respuesta como la siguiente:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

5. Verifique la respuesta.

```
cat response.txt
```

Debería ver una respuesta como la siguiente:

```
Echoing request: '{"text":"Hello"}'
```

## Actualización del tiempo de ejecución

1. Para registrar información sobre el entorno de ejecución, actualice el script de tiempo de ejecución para emitir variables de entorno.

### Example bootstrap

```
#!/bin/sh

set -euo pipefail

# Configure runtime to output environment variables
echo "## Environment variables:"
env

# Load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event. The HTTP request will block until one is received
    EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

    # Extract request ID by scraping response headers received above
    REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
'[:space:]' | cut -d: -f2)

    # Run the handler function from the script
    RESPONSE=$((echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

    # Send the response
    curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "$RESPONSE"
done
```

2. Cree un archivo .zip que contenga la nueva versión del archivo bootstrap.

```
zip runtime.zip bootstrap
```

3. Cree una nueva versión de la capa bash-runtime.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://runtime.zip
```

4. Configure la función para utilizar la nueva versión de la capa.

```
aws lambda update-function-configuration --function-name bash-runtime \--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

## Uso compartido de la capa

Para compartir una función con otra Cuenta de AWS, agregue una declaración de permisos entre cuentas a la [política basada en los recursos](#) de la capa. Ejecute el comando [add-layer-version-permission](#) y especifique el ID de la cuenta como principal. En cada instrucción, puede conceder permiso a una única cuenta, a todas las cuentas o a una organización en [AWS Organizations](#).

El siguiente ejemplo concede a la cuenta 111122223333 acceso a la versión 2 de la capa bash-runtime.

```
aws lambda add-layer-version-permission \--layer-name bash-runtime \--version-number 2 \--statement-id xaccount \--action lambda:GetLayerVersion \--principal 111122223333 \--output text
```

Debería ver una salida similar a esta:

```
{"Sid":"xaccount","Effect":"Allow","Principal":{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:us-east-1:123456789012:layer:bash-runtime:2"}
```

Los permisos solo se aplican a una única versión de una capa. Repita el proceso cada vez que cree una nueva versión de la capa.

## Limpieza

Elimine cada versión de la capa.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Puesto que la función contiene una referencia a la versión 2 de la capa, esta sigue existiendo en Lambda. Si bien la función sigue funcionando, ya no se pueden configurar funciones para que utilicen la versión eliminada. Si modifica la lista de capas de la función, deberá especificar una nueva versión u omitir la capa eliminada.

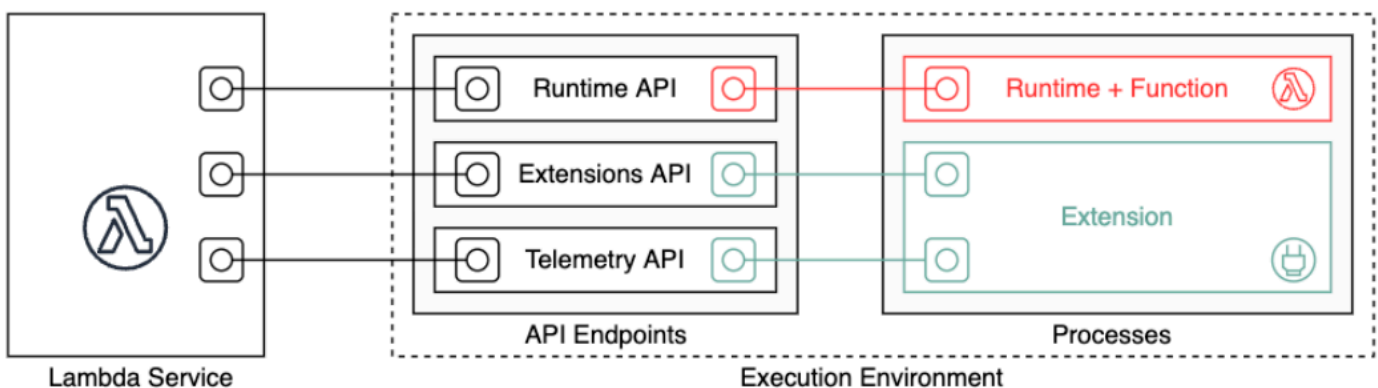
Elimine la función con el comando [delete-function](#).

```
aws lambda delete-function --function-name bash-runtime
```

# Comprender el ciclo de vida del entorno de ejecución de Lambda

Lambda invoca la función en un entorno de ejecución, que proporciona un entorno en tiempo de ejecución seguro y aislado. El entorno de ejecución administra los recursos necesarios para ejecutar la función. El entorno de ejecución también proporciona compatibilidad del ciclo de vida para el tiempo de ejecución de la función y cualquier [extensión externa](#) asociada a la función.

El tiempo de ejecución de la función de Lambda se comunica con la [API de tiempo de ejecución](#). Las extensiones se comunican con Lambda mediante la [API de extensiones](#). Las extensiones también pueden recibir mensajes de registro y otros datos de telemetría de la función mediante la [API de telemetría](#).



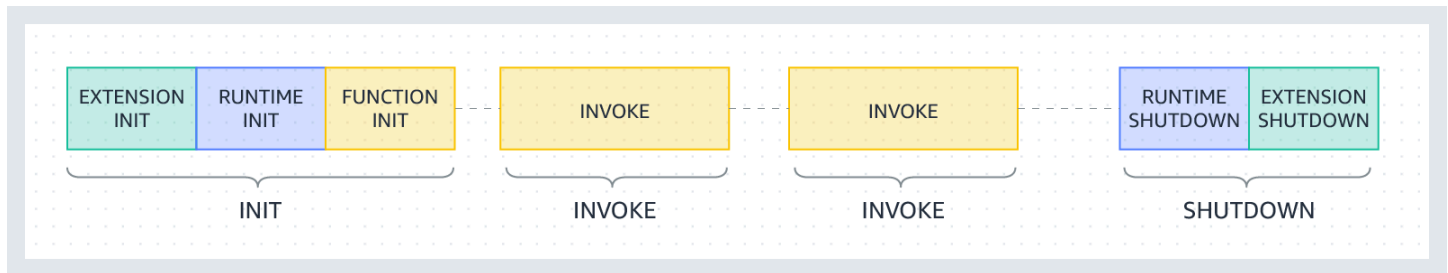
Al crear una función de Lambda, se debe especificar información de configuración, como la cantidad de memoria y el tiempo máximo de ejecución asignados a su función. Lambda utiliza esta información para configurar el entorno de ejecución.

El tiempo de ejecución de la función y cada extensión externa son procesos que se ejecutan dentro del entorno de ejecución. Los permisos, recursos, credenciales y variables de entorno se comparten entre la función y las extensiones.

## Temas

- [Ciclo de vida del entorno de ejecución de Lambda](#)
- [Implementación de la ausencia de estado en las funciones](#)

# Ciclo de vida del entorno de ejecución de Lambda



Cada fase comienza con un evento que Lambda envía al tiempo de ejecución y a todas las extensiones registradas. El tiempo de ejecución y cada extensión registrada indica la finalización mediante el envío de una solicitud API Next. Lambda congela el entorno de ejecución cuando el tiempo de ejecución y cada extensión se han completado y no hay eventos pendientes.

## Temas

- [Fase "init"](#)
- [Errores durante la fase Init](#)
- [Fase Restore \(solo Lambda SnapStart\)](#)
- [Fase "invoke"](#)
- [Errores durante la fase Invoke](#)
- [Fase "shutdown"](#)

## Fase "init"

En la fase Init, Lambda realiza tres tareas:

- Comenzar todas las extensiones (`Extension init`)
- Bootstrap del tiempo de ejecución (`Runtime init`)
- Ejecutar el código estático de la función (`Function init`)
- Ejecute cualquier [enlace en tiempo de ejecución](#) `beforeCheckpoint` (solo Lambda SnapStart)

La fase Init finaliza cuando el tiempo de ejecución y todas las extensiones señalan que están listas mediante el envío de una solicitud Next a la API. La fase Init está limitada a 10 segundos. Si las tres tareas no se completan en 10 segundos, Lambda vuelve a intentar la fase Init en el momento de la primera invocación de la función con el tiempo de espera de la función configurado.

Si [Lambda SnapStart](#) está activada, la fase `Init` ocurre cuando publica una versión de la función. Lambda guarda una instantánea del estado de la memoria y del disco del entorno de ejecución iniciado, conserva la instantánea cifrada y la almacena en caché para acceder a ella con baja latencia. Si tiene un [enlace de tiempo de ejecución](#) `beforeCheckpoint`, el código se ejecuta al final de la fase `Init`.

### Note

El tiempo de espera de 10 segundos no se aplica a las funciones que utilizan la simultaneidad aprovisionada o SnapStart. Para la simultaneidad aprovisionada y las funciones SnapStart, el código de inicialización puede ejecutarse durante un máximo de 15 minutos. El límite de tiempo es de 130 segundos o el tiempo de espera de la función configurado (máximo de 900 segundos), lo que sea mayor.

Cuando utiliza la [simultaneidad aprovisionada](#), Lambda inicializa el entorno de ejecución al configurar los ajustes del equipo para una función. Lambda también garantiza que los entornos de ejecución inicializados se encuentren siempre disponibles antes de las invocaciones. Es posible que vea intervalos entre las fases de inicialización e invocación de la función. Dependiendo del tiempo de ejecución y la configuración de memoria de su función, es posible que también vea variabilidad de latencia en la primera invocación en un entorno de ejecución inicializado.

En el caso de las funciones que utilizan la simultaneidad bajo demanda, Lambda puede, en ocasiones, inicializar los entornos de ejecución antes de las solicitudes de invocación. Cuando esto ocurre, también puede observar un intervalo de tiempo entre las fases de inicialización e invocación de la función. Se recomienda que no dependa de este comportamiento.

## Errores durante la fase `Init`

Si una función se bloquea o agota el tiempo de espera durante la fase `Init`, Lambda emite la información sobre el error en el registro `INIT_REPORT`.

Example — Registro `INIT_REPORT` para el tiempo de espera

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: timeout
```



## Example — Registro INIT\_REPORT para un error en la extensión

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: error Error Type:
Extension.Crash
```

Si la fase `Init` se completa correctamente, Lambda no emite el registro `INIT_REPORT`, a menos que se active `SnapStart`. Las funciones de `SnapStart` siempre emiten `INIT_REPORT`. Para obtener más información, consulte [Monitoreo para Lambda SnapStart](#).

## Fase Restore (solo Lambda SnapStart)

Cuando se invoca por primera vez una función `SnapStart` y, a medida que la función escala, Lambda vuelve a activar los nuevos entornos de ejecución a partir de la instantánea conservada, en lugar de activar la función desde cero. Si tiene un [enlace de tiempo de ejecución](#) `afterRestore()`, el código se ejecuta al final de la fase `Restore`. Se le cobrará por la duración de los enlaces de tiempo de ejecución `afterRestore()`. El tiempo de ejecución (JVM) debe cargarse, y los enlaces del tiempo de ejecución `afterRestore()` deben completarse antes de que transcurra el tiempo de espera (10 segundos). De lo contrario, obtendrá una excepción `SnapStartTimeoutException`. Cuando se completa la fase `Restore`, Lambda invoca el controlador de funciones (la fase ["invoke"](#)).

## Errores durante la fase de Restauración

Si la fase `Restore` falla, Lambda emite la información sobre el error en el registro `RESTORE_REPORT`.

### Example — Registro RESTORE\_REPORT para el tiempo de espera

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: timeout
```

### Example — Registro RESTORE\_REPORT para errores en el enlace del tiempo de ejecución

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: error Error Type: Runtime.ExitError
```

Para obtener más información sobre el registro `RESTORE_REPORT`, consulte [Monitoreo para Lambda SnapStart](#).

## Fase "invoke"

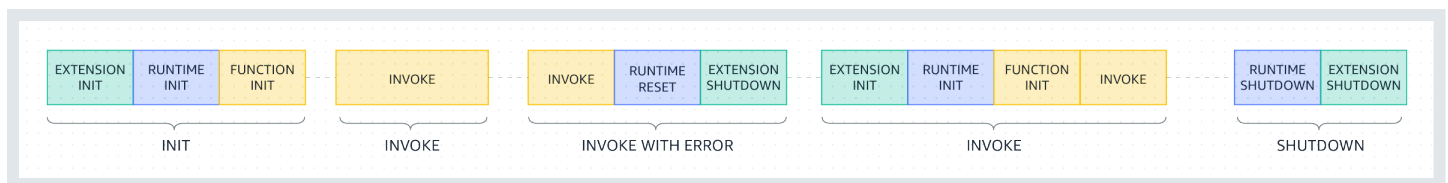
Cuando se invoca una función de Lambda en respuesta a una solicitud API Next, Lambda envía un evento `Invoke` al tiempo de ejecución y a cada extensión.

La configuración de tiempo de espera de la función limita la duración de toda la fase Invoke. Por ejemplo, si establece el tiempo de espera de la función en 360 segundos, la función y todas las extensiones deben completarse en 360 segundos. Tenga en cuenta que no hay una fase posterior a "invoke" independiente. La duración es la suma de todo el tiempo de invocación (tiempo de ejecución + extensiones) y no se calcula hasta que la función y todas las extensiones han terminado la ejecución.

La fase "invoke" finaliza después de que el tiempo de ejecución y todas las extensiones indiquen que se han terminado mediante el envío de una solicitud Next a la API.

## Errores durante la fase Invoke

Si la función Lambda se bloquea o agota el tiempo de espera durante la fase Invoke, Lambda restablece el entorno de ejecución. El siguiente diagrama ilustra el comportamiento del entorno de ejecución de Lambda cuando se produce un error de invocación:



En el diagrama anterior:

- La primera fase es la fase INIT, que se ejecuta sin errores.
- La segunda fase es la fase INVOKE, que se ejecuta sin errores.
- En algún punto, supongamos que la función tiene un error de invocación (como un error de tiempo de espera de función o de tiempo de ejecución). La tercera fase, denominada INVOKE WITH ERROR, ilustra este escenario. Cuando esto sucede, el servicio de Lambda efectúa un reinicio. El restablecimiento se comporta como un evento Shutdown. Primero, Lambda apaga el tiempo de ejecución, luego envía un evento Shutdown a cada extensión externa registrada. El evento incluye el motivo del apagado. Si este entorno se utiliza para una nueva invocación, Lambda reinicia la extensión y el tiempo de ejecución juntos como parte de la siguiente invocación.

Tenga en cuenta que el restablecimiento de Lambda no borra el contenido del directorio /tmp antes de la siguiente fase Init. Este comportamiento es coherente con la fase de apagado regular.

**Note**

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastreo emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

Si la configuración de registro del sistema de su función está configurada como texto sin formato, este cambio afectará a los mensajes de registro capturados en CloudWatch Logs cuando su función experimente un error de invocación. En los siguientes ejemplos, se muestran los resultados de los registros en formatos antiguos y nuevos.

Estos cambios se implementarán en las próximas semanas, y todas las funciones en todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastreo.

Example Resultado del Registro de CloudWatch (bloqueo de la extensión o el tiempo de ejecución): estilo antiguo

```
START RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Version: $LATEST
RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Error: Runtime exited without providing a reason
Runtime.ExitError
END RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1
REPORT RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Duration: 933.59 ms Billed Duration: 934 ms Memory Size: 128 MB Max Memory Used: 9 MB
```

Example Resultado del Registro de CloudWatch (se acabó el tiempo de espera de la función): estilo antiguo

```
START RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Version: $LATEST
2024-03-04T17:22:38.033Z b70435cc-261c-4438-b9b6-efe4c8f04b21 Task timed out after 3.00 seconds
END RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21
REPORT RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Duration: 3004.92 ms Billed Duration: 3000 ms Memory Size: 128 MB Max Memory Used: 33 MB Init Duration: 111.23 ms
```

El nuevo formato de los registros de CloudWatch incluye un campo `status` adicional en la línea `REPORT`. En caso de que se bloquee la extensión o el tiempo de ejecución, la línea `REPORT` también incluye un campo `ErrorType`.

Example Resultado del Registro de CloudWatch (bloqueo de la extensión o el tiempo de ejecución): estilo nuevo

```
START RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Version: $LATEST
END RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd
REPORT RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Duration: 133.61 ms Billed
Duration: 133 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 80.00
ms Status: error Error Type: Runtime.ExitError
```

Example Resultado del Registro de CloudWatch (se acabó el tiempo de espera de la función): estilo nuevo

```
START RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Version: $LATEST
END RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda
REPORT RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Duration: 3016.78 ms Billed
Duration: 3016 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 84.00
ms Status: timeout
```

- La cuarta fase representa la fase de `INVOKE` que sigue de inmediato a un error de invocación. Aquí, Lambda vuelve a inicializar el entorno al volver a ejecutar la fase `INIT`. Esto se denomina inicio suprimido. Cuando se producen inicializaciones suprimidas, Lambda no informa explícitamente de una fase `INIT` adicional en Registros de CloudWatch. En cambio, puede observar que la duración de la línea `INFORME` incluye una duración `INIT` adicional + la duración de `INVOKE`. Por ejemplo, supongamos que visualiza los siguientes registros en CloudWatch:

```
2022-12-20T01:00:00.000-08:00 START RequestId: XXX Version: $LATEST
2022-12-20T01:00:02.500-08:00 END RequestId: XXX
2022-12-20T01:00:02.500-08:00 REPORT RequestId: XXX Duration: 3022.91 ms
Billed Duration: 3000 ms Memory Size: 512 MB Max Memory Used: 157 MB
```

En este ejemplo, la diferencia entre las marcas de tiempo de `INFORME` e `INICIO` es de 2,5 segundos. Esto no coincide con la duración reportada de 3022,91 milisegundos, porque no tiene en cuenta el tiempo `INIT` adicional (inicio suprimido) que realizó Lambda. En este ejemplo, puede inferir que la fase de `INVOKE` real tardó 2,5 segundos.

Para obtener más información sobre este comportamiento, puede utilizar la [Acceso a datos de telemetría en tiempo real para extensiones mediante la API de telemetría](#). La API de telemetría emite eventos de INIT\_START, INIT\_RUNTIME\_DONE y INIT\_REPORT con phase=invoke siempre que se supriman inicializaciones entre las fases de invocación.

- La quinta fase representa la fase SHUTDOWN, que se ejecuta sin errores.

## Fase "shutdown"

Cuando Lambda está a punto de cerrar el tiempo de ejecución, envía un evento Shutdown a cada extensión externa registrada. Las extensiones pueden utilizar este tiempo para las tareas de limpieza finales. El evento Shutdown es una respuesta a una solicitud Next a la API.

Duración: toda la fase Shutdown está limitada a 2 segundos. Si el tiempo de ejecución o cualquier extensión no responde, Lambda lo termina a través de una señal (SIGKILL).

Después de que la función y todas las extensiones se hayan completado, Lambda mantiene el entorno de ejecución durante algún tiempo en previsión de otra invocación de función. Sin embargo, Lambda finaliza los entornos de ejecución cada pocas horas para permitir las actualizaciones y el mantenimiento del tiempo de ejecución, incluso para las funciones que se invocan de forma continua. No debe suponer que el entorno de ejecución durará de manera indefinida. Para obtener más información, consulte [Implementación de la ausencia de estado en las funciones](#).

Cuando se invoca de nuevo la función, Lambda descongela el entorno para su reutilización. La reutilización del entorno de ejecución tiene las siguientes implicaciones:

- Los objetos declarados fuera del método del controlador de la función permanecen inicializados, lo que proporciona una optimización adicional cuando la función se invoca de nuevo. Por ejemplo, si la función de Lambda establece una conexión con una base de datos, en lugar de volver a establecer la conexión, se utiliza la conexión original en posteriores invocaciones. Le recomendamos que agregue lógica al código para comprobar si existe una conexión antes de crear una nueva.
- Cada entorno de ejecución proporciona entre 512 MB y 10 240 MB en incrementos de 1 MB de espacio en disco en el directorio de /tmp. El contenido del directorio se conserva al congelar el entorno de ejecución, proporcionando una caché transitoria que se puede utilizar para varias invocaciones. Puede agregar código adicional para comprobar si la caché dispone de los datos que se almacenaron. Para obtener más información sobre los límites de tamaño de implementación, consulte [Cuotas de Lambda](#).

- Los procesos en segundo plano o devoluciones de llamada iniciados por la función de Lambda y no completados cuando la función finalizó se reanudan si reutiliza el entorno de ejecución. Asegúrese de que los procesos en segundo plano o las devoluciones de llamada del código se completen antes de que este finalice.

## Implementación de la ausencia de estado en las funciones

Al escribir el código de la función de Lambda, trate el entorno de ejecución como si existiera ausencia de estado, al suponer que solo existe para una única invocación. Lambda finaliza los entornos de ejecución cada pocas horas para permitir las actualizaciones y el mantenimiento del tiempo de ejecución, incluso para las funciones que se invocan de forma continua. Inicialice cualquier estado necesario (por ejemplo, la búsqueda de un carrito de compras de una tabla de Amazon DynamoDB) cuando se inicie la función. Antes de su aparición, confirme los cambios permanentes en los datos para los almacenamientos duraderos, tales como Amazon Simple Storage Service (Amazon S3), DynamoDB, o Amazon Simple Queue Service (Amazon SQS). Evite confiar en las estructuras de datos existentes, los archivos temporales o los estados que abarquen las invocaciones, como los contadores o los agregados. Esto garantiza que la función gestione cada invocación de forma independiente.

# Configuración de funciones de AWS Lambda

Obtenga información sobre cómo configurar las capacidades y las opciones principales de su función de Lambda mediante la API o la consola de Lambda.

## [Memoria](#)

Obtenga información sobre cómo y cuándo aumentar la memoria de una función.

## [Almacenamiento efímero](#)

Obtenga información sobre cómo y cuándo aumentar la capacidad de almacenamiento temporal de la función.

## [Timeout \(Tiempo de espera\)](#)

Obtenga información sobre cómo y cuándo aumentar el valor del tiempo de espera de la función.

## [Variables de entorno](#)

Puede hacer que el código de su función sea portátil y mantener los secretos fuera de su código almacenándolos en la configuración de la función mediante las variables de entorno.

## [Redes salientes](#)

Puede utilizar la función de Lambda con los recursos de AWS de una nube de Amazon VPC. La conexión de la función a una VPC le permite obtener acceso a los recursos de una subred privada, como bases de datos relacionales y cachés.

## [Redes entrantes](#)

Puede utilizar un punto de conexión de VPC de tipo interfaz para invocar las funciones de Lambda sin pasar por la red pública de Internet.

## [Sistema de archivos](#)

Puede usar la función de Lambda para montar un sistema de Amazon EFS en un directorio local. Un sistema de archivos permite al código de su función acceder a los recursos compartidos de forma segura y en alta simultaneidad, y también le permite modificarlos.

## [Alias](#)

Puede configurar sus clientes para invocar una versión específica de la función de Lambda mediante un alias en lugar de actualizar el cliente.

## Versiones

Al publicar una versión de su función, puede almacenar el código y la configuración como recursos independientes que no pueden modificarse.

## Etiquetas

Utilice etiquetas para activar el control de acceso basado en atributos (ABAC), organizar las funciones de Lambda y filtrar y generar informes sobre las funciones mediante los servicios de Administración de costos y facturación de AWS Cost Explorer o AWS.

## Transmisión de respuestas

Puede configurar las URL de función de Lambda para devolver las cargas de respuesta a los clientes. La transmisión de respuestas puede beneficiar a las aplicaciones sensibles a la latencia al mejorar el rendimiento del tiempo hasta el primer byte (TTFB). Esto se debe a que puede volver respuestas parciales al cliente a medida que estén disponibles. Además, puede usar la transmisión de respuestas para crear funciones que devuelvan cargas más grandes.



# Implementación de funciones de Lambda como archivos .zip

Cuando se crea una función Lambda, empaquete el código de función en un paquete de implementación. Lambda admite dos tipos de paquetes de implementación: imágenes de contenedor y archivos .zip. El flujo de trabajo para crear una función depende del tipo de paquete de implementación. Para configurar una función definida como una imagen de contenedor, consulte [the section called “Imágenes de contenedor”](#).

Puede utilizar la consola de Lambda y la API de Lambda para crear una función definida con un archivo .zip. También puede cargar un archivo .zip actualizado para cambiar el código de función.

## Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

## Temas

- [Creación de la función de Lambda](#)
- [Uso del editor de código de la consola](#)
- [Actualización del código de la función](#)
- [Cambio del tiempo de ejecución](#)
- [Modificación de la arquitectura](#)
- [Uso de la API de Lambda](#)
- [AWS CloudFormation](#)

## Creación de la función de Lambda

Cuando se crea una función definida con un archivo .zip, se elige una plantilla de código, la versión de idioma y el rol de ejecución para la función. Agregue su código de función después de que Lambda cree la función.

### Cómo crear la función

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.

2. Elija Create function (Crear función).
3. Seleccione Author from scratch (Crear desde cero) o Use a blueprint (Usar proyecto) para crear su función.
4. Bajo Basic information (Información básica), haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función. Los nombres de las funciones están limitados a 64 caracteres de longitud.
  - b. Para Runtime (Tiempo de ejecución), elija la versión del idioma que desea utilizar para su función.
  - c. (Opcional) Para Architecture (Arquitectura), elija la arquitectura del conjunto de instrucciones que utilizará para su función. La arquitectura predeterminada es x86\_64. Cuando cree el paquete de implementación para su función, asegúrese de que sea compatible con esta [arquitectura del conjunto de instrucciones](#).
5. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o crear un rol existente.
6. (Opcional) Expanda Advanced settings (Configuración avanzada). Puede elegir una Configuración de firma de código para la función. También puede configurar una (Amazon VPC) para que tenga acceso a la función.
7. Elija Create function (Crear función).

Lambda crea la nueva función. Ahora puede usar la consola para agregar el código de función y configurar otros parámetros y características de función. Para obtener instrucciones de implementación de código, consulte la página del controlador para conocer el tiempo de ejecución que utiliza la función.

Node.js

[Implementar funciones Node.js de Lambda con archivos de archivo.zip](#)

Python

[Uso de archivos .zip para funciones de Lambda en Python](#)

Ruby

[Implementar funciones de Lambda de Ruby con archivos .zip](#)

Java

[Implementar funciones de Lambda Java con archivos de archivo .zip o JAR](#)

## Go

[Implementar funciones de Lambda en Go con archivos .zip](#)

## C#

[Crear e implementar funciones de Lambda C# con archivos de archivo .zip](#)

## PowerShell

[Implemente funciones PowerShell Lambda con archivos de archivos.zip](#)

## Uso del editor de código de la consola

La consola crea una función de Lambda con un único archivo de fuente. Para los lenguajes de scripting, puede editar este archivo y agregar más archivos con el editor de código integrado. Para guardar los cambios, elija Guardar. A continuación, para ejecutar el código, elija Pruebas.

### Note

La consola de Lambda utiliza AWS Cloud9 para proporcionar un entorno de desarrollo integrado en el navegador. También puede utilizar AWS Cloud9 para desarrollar funciones de Lambda en su propio entorno. Para obtener más información, consulte [Trabajar con funciones de AWS Lambda utilizando AWS Toolkit](#) en la guía del usuario de AWS Cloud9.

Al guardar el código de función, la consola de Lambda crea un paquete de implementación de archivo .zip. Cuando desarrolle el código de función fuera de la consola (mediante un IDE), debe [crear un paquete de implementación](#) para cargar el código a la función de Lambda.

## Actualización del código de la función

Para los lenguajes de scripting (Node.js, Python y Ruby), puede editar el código de la función en el editor de código integrado. Si el tamaño del código supera los 3 MB, o si necesita agregar bibliotecas, o para lenguajes incompatibles con el editor (Java, Go, C#), debe cargar el código de función como un archivo .zip. Si el archivo .zip tiene un tamaño inferior a los 50 MB, puede cargarlo desde su equipo local. Si el tamaño del archivo supera los 50 MB, cárguelo a la función desde un bucket de Amazon S3.

Para cargar código de función como un archivo .zip

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función que desea actualizar y elija la pestaña Código.
3. En Fuente de código, seleccione Cargar desde.
4. Elija .zip file (Archivo .zip) y, a continuación, elija Upload (Cargar).
  - En el selector de archivos, seleccione la versión de imagen nueva y elija Open (Abrir) y, luego, Save (Guardar).
5. (Alternativa al paso 4) Elija Amazon S3 location (Ubicación de Amazon S3).
  - En el cuadro de texto, introduzca el vínculo URL de S3 del archivo .zip y, después, elija Save (Guardar).

## Cambio del tiempo de ejecución

Si actualiza la configuración de la función para utilizar un nuevo tiempo de ejecución, es posible que deba actualizar el código de la función para que sea compatible con el tiempo de ejecución nuevo. Si actualiza la configuración de la función para utilizar un tiempo de ejecución diferente, debe proporcionar un nuevo código de función que sea compatible con el tiempo de ejecución y la arquitectura. Para obtener instrucciones acerca de cómo crear un paquete de implementación para el código de función, consulte la página del controlador para conocer el tiempo de ejecución que utiliza la función.

Las imágenes base de Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3 y versiones posteriores se basan en la imagen de contenedor mínima de Amazon Linux 2023. Las imágenes base anteriores utilizan Amazon Linux 2. AL2023 ofrece varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como glibc. Para obtener más información, consulte [Presentación del tiempo de ejecución de Amazon Linux 2023 para AWS Lambda](#) en el Blog de informática de AWS.

Para cambiar el tiempo de ejecución

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función que desea actualizar y elija la pestaña Código.
3. Desplácese hasta la sección Configuración de tiempo de ejecución, que se encuentra en el editor de código.

4. Elija Editar.
  - a. Para Runtime (Tiempo de ejecución), seleccione el identificador del tiempo de ejecución.
  - b. Para Handler (Controlador), especifique el nombre de archivo y el controlador de su función.
  - c. Para Architecture (Arquitectura), elija la arquitectura del conjunto de instrucciones que utilizará para su función.
5. Seleccione Guardar.

## Modificación de la arquitectura

Antes de cambiar la arquitectura del conjunto de instrucciones, debe asegurarse de que el código de función sea compatible con la arquitectura de destino.

Si utiliza Node.js, Python o Ruby y edita el código de función en el editor integrado, el código existente puede ejecutarse sin modificaciones.

Sin embargo, si proporciona el código de función mediante un paquete de implementación de archivo .zip, debe preparar un nuevo archivo .zip recopilado y creado de forma correcta para el tiempo de ejecución de destino y la arquitectura del conjunto de instrucciones. Para obtener instrucciones, consulte la página del controlador para conocer el tiempo de ejecución de la función.

Para cambiar la arquitectura del conjunto de instrucciones

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función a actualizar y luego elija la pestaña Code (Código).
3. En Runtime settings (Configuración de tiempo de ejecución), elija Edit (Editar).
4. Para Architecture (Arquitectura), elija la arquitectura del conjunto de instrucciones que utilizará para su función.
5. Seleccione Save.

## Uso de la API de Lambda

Para crear y configurar una función que utilice un archivo .zip, utilice las siguientes operaciones de la API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

- [UpdateFunctionConfiguration](#)

## AWS CloudFormation

Puede usar AWS CloudFormation para crear una función Lambda con archivos .zip. En su plantilla AWS CloudFormation, el recurso `AWS::Lambda::Function` especifica la función Lambda. Para obtener descripciones de las propiedades del recurso `AWS::Lambda::Function`, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

En el recurso `AWS::Lambda::Function`, establezca las siguientes propiedades para crear una función definida como un archivo .zip:

- `AWS::Lambda::Function`
  - `PackageType`: establezca en `Zip`.
  - `Código`: introduzca el nombre del bucket de Amazon S3 y el nombre del archivo .zip en los campos `S3Bucket` y `S3Key`. Para Node.js o Python, puede proporcionar código fuente en línea de su función Lambda.
  - `Tiempo de ejecución`: establece el valor de tiempo de ejecución.
  - `Arquitectura`: establezca el valor de la arquitectura en `arm64` para utilizar el procesador Graviton2 de AWS. De forma predeterminada, el valor de la arquitectura es `x86_64`.

# Crear una función de Lambda con una imagen de contenedor

El código de la función AWS Lambda se compone de scripts o programas compilados y sus dependencias. Utiliza un paquete de implementación para implementar su código de función en Lambda. Lambda admite dos tipos de paquetes de implementación: imágenes de contenedor y archivos .zip.

Hay tres formas de crear una imagen de contenedor para una función de Lambda:

- [Uso de una imagen base de AWS para Lambda](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir en ella un [cliente de interfaz de tiempo de ejecución](#) para su lenguaje.

- [Uso de una imagen base que no es de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir en ella un [cliente de interfaz de tiempo de ejecución](#) para su lenguaje.

## Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedor eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

Para crear una función de Lambda a partir de una imagen de contenedor, cree su imagen de manera local y cárguela en un repositorio de Amazon Elastic Container Registry (Amazon ECR). A continuación, especifique el URI del repositorio cuando cree la función. El repositorio Amazon ECR debe estar en el mismo Región de AWS que la función de Lambda. Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

En esta página, se explican los tipos de imágenes base y los requisitos para crear imágenes de contenedor compatibles con Lambda.

### Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

## Temas

- [Requisitos](#)
- [Uso de una imagen base de AWS para Lambda](#)
- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)
- [Uso de una imagen base que no es de AWS](#)
- [Clientes de interfaz de tiempo de ejecución](#)
- [Permisos de Amazon ECR](#)
- [Ciclo de vida de la función](#)

## Requisitos

Instale la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#) y la [CLI de Docker](#). Además, tenga en cuenta los siguientes requisitos:

- La imagen de contenedor debe implementar la [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#). Los [clientes de interfaz de tiempo de ejecución](#) de AWS de código abierto implementan la API. Puede agregar un cliente de interfaz de tiempo de ejecución a su imagen base preferida para que sea compatible con Lambda.



- La imagen de contenedor debe poder ejecutarse en un sistema de archivos de solo lectura. Su código de función puede acceder a un directorio `/tmp` con escritura permitida con entre 512 MB y 10,240 MB, en incrementos de 1 MB, de almacenamiento.
- El usuario predeterminado de Lambda debe ser capaz de leer todos los archivos necesarios para ejecutar el código de función. Lambda sigue las prácticas recomendadas de seguridad definiendo un usuario predeterminado de Linux con permisos de privilegio mínimo. Compruebe que el código de su aplicación no depende de los archivos que otros usuarios de Linux tienen restricciones para ejecutar.
- Lambda solo admite imágenes de contenedor basadas en Linux.
- Lambda proporciona imágenes base de varias arquitecturas. Sin embargo, la imagen que crea para su función debe tener como destino a solo una de las arquitecturas. Lambda no admite funciones que utilizan imágenes de contenedor de varias arquitecturas.

## Uso de una imagen base de AWS para Lambda

Puede utilizar una de las [imágenes base de AWS](#) para Lambda a fin de crear la imagen de contenedor para el código de función. Las imágenes base están precargadas con un tiempo de ejecución de lenguaje y otros componentes necesarios para ejecutar una imagen contenedor en Lambda. Agregue su código de función y dependencias a la imagen base y luego lo empaqueta como una imagen de contenedor.

AWS proporciona periódicamente actualizaciones a las imágenes base AWS para Lambda. Si su Dockerfile incluye el nombre de la imagen en la propiedad FROM, el cliente de Docker extrae la última versión de la imagen del [repositorio de Amazon ECR](#). Para utilizar la imagen base actualizada, debe reconstruir la imagen contenedor y [actualizar el código de función](#).

Las imágenes base de Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3 y versiones posteriores se basan en la [imagen de contenedor mínima de Amazon Linux 2023](#). Las imágenes base anteriores utilizan Amazon Linux 2. AL2023 ofrece varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como `glibc`.

Las imágenes basadas en AL2023 utilizan `microdnf` (enlazadas simbólicamente como `dnf`) como administrador de paquetes en lugar de `yum`, que es el administrador de paquetes predeterminado en Amazon Linux 2. `microdnf` es una implementación independiente de `dnf`. Para obtener una lista de los paquetes que se incluyen en las imágenes basadas en AL2023, consulte las columnas de Minimal Container de [Comparing packages installed on Amazon Linux 2023 Container Images](#). Para

obtener más información sobre las diferencias entre AL2023 y Amazon Linux 2, consulte [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) en el Blog de informática de AWS.

### Note

Para ejecutar imágenes basadas en AL2023 de forma local, incluso con AWS Serverless Application Model (AWS SAM), debe usar Docker en la versión 20.10.10 o posterior.

Para crear una imagen de contenedor a partir de una imagen base de AWS, elija las instrucciones para el lenguaje que prefiera:

- [Node.js](#)
- [TypeScript](#) (usa una imagen base de Node.js)
- [Python](#)
- [Java](#)
- [Go](#)
- [.NET](#)
- [Ruby](#)

## Uso de una imagen base exclusiva del sistema operativo de AWS

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir en ella un [cliente de interfaz de tiempo de ejecución](#) para su lenguaje.

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
al2023	Tiempo de ejecución exclusivo	Amazon Linux 2023	<a href="#">Dockerfile para tiempo de ejecución exclusivo</a>	

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
	del sistema operativo		<a href="#">del sistema operativo en GitHub</a>	
al2	Tiempo de ejecución exclusivo del sistema operativo	Amazon Linux 2	<a href="#">Dockerfile para tiempo de ejecución exclusivo del sistema operativo en GitHub</a>	

Galería pública de Amazon Elastic Container Registry: [gallery.ecr.aws/lambda/provided](https://gallery.ecr.aws/lambda/provided)

## Uso de una imagen base que no es de AWS

Lambda admite cualquier imagen que se ajuste a uno de los siguientes formatos de manifiesto de imagen:

- Docker Image Manifest V2 Schema 2 (usado con Docker versión 1.10 y posteriores)
- Especificaciones de la iniciativa de contenedores abiertos (OCI) (versión 1.0.0 y posteriores)

Lambda admite un tamaño máximo de imagen sin comprimir de 10 GB, incluidas todas las capas.

### Note

Para que la imagen sea compatible con Lambda, debe incluir en ella un [cliente de interfaz de tiempo de ejecución](#) para su lenguaje.

## Cientes de interfaz de tiempo de ejecución

Si utiliza una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir un cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución debe extender la [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de función. AWS proporciona clientes de interfaz de tiempo de ejecución de código abierto para los siguientes lenguajes:

- [Node.js](#)
- [Python](#)
- [Java](#)
- [.NET](#)
- [Go](#)
- [Ruby](#)
- [Rust](#): el [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y se utiliza solo con fines de evaluación.

Si utiliza un lenguaje que no tiene un cliente de interfaz de tiempo de ejecución de AWS, debe crear el suyo propio.

## Permisos de Amazon ECR

Antes de crear la función de Lambda desde una imagen de contenedor, debe crear la imagen de forma local y cargarla en un repositorio de Amazon ECR. Cuando cree la función, especifique el URI del repositorio de Amazon ECR.

Asegúrese de que los permisos para el usuario o el rol que crean la función incluyan `GetRepositoryPolicy` y `SetRepositoryPolicy`.

Por ejemplo, utilice la consola de IAM para crear un rol con la siguiente política:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
  ]
}
```

## Políticas de repositorios de Amazon ECR

Para una función en la misma cuenta que la imagen de contenedor en Amazon ECR, puede agregar los permisos `ecr:BatchGetImage` y `ecr:GetDownloadUrlForLayer` a su política de repositorios de Amazon ECR. En el siguiente ejemplo se muestra la política mínima:

```
{
  "Sid": "LambdaECRImageRetrievalPolicy",
  "Effect": "Allow",
  "Principal": {
    "Service": "lambda.amazonaws.com"
  },
  "Action": [
    "ecr:BatchGetImage",
    "ecr:GetDownloadUrlForLayer"
  ]
}
```

Para obtener más información sobre los permisos de repositorio de Amazon ECR, consulte [Políticas de repositorio privado](#) en la Guía del usuario de Amazon Elastic Container Registry.

Si el repositorio de Amazon ECR no incluye estos permisos, Lambda agrega `ecr:BatchGetImage` y `ecr:GetDownloadUrlForLayer` a los permisos del repositorio de imágenes de contenedor. Lambda puede agregar estos permisos solo si la entidad principal que llama a Lambda tiene los permisos `ecr:getRepositoryPolicy` y `ecr:setRepositoryPolicy`.

Para ver o editar los permisos del repositorio de Amazon ECR, siga las instrucciones de [Establecer una instrucción de política de repositorio privado](#) en la Guía del usuario de Amazon Elastic Container Registry.

### Permisos entre cuentas de Amazon ECR

Una cuenta diferente en la misma región puede crear una función que utiliza una imagen de contenedor propiedad de su cuenta. En el siguiente ejemplo, su [política de permisos de repositorio de Amazon ECR](#) necesita las siguientes instrucciones para otorgar acceso al número de cuenta 123456789012.

- `CrossAccountPermission`: permite a la cuenta 123456789012 crear y actualizar funciones Lambda que utilizan imágenes de este repositorio ECR.
- `LambdaECRImageCrossAccountRetrievalPolicy`: con el tiempo, Lambda establecerá el estado de una función como inactiva si esta no se invoca durante un período prolongado. Esta instrucción

es necesaria para que Lambda pueda recuperar la imagen de contenedor para su optimización y almacenamiento en caché en nombre de la función propiedad de 123456789012.

### Example — Agregar permiso entre cuentas al repositorio

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPermission",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      }
    },
    {
      "Sid": "LambdaECRIImageCrossAccountRetrievalPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "StringLike": {
          "aws:sourceARN": "arn:aws:lambda:us-east-1:123456789012:function:*"
        }
      }
    }
  ]
}
```

Para brindar acceso a varias cuentas, agregue los ID de cuenta a la lista de la entidad principal en la política `CrossAccountPermission` y a la lista de evaluación de condiciones en `LambdaECRIImageCrossAccountRetrievalPolicy`.

Si está trabajando con varias cuentas en una organización de AWS, le recomendamos que enumere cada ID de cuenta en la política de permisos ECR. Este enfoque se alinea con la práctica recomendada de seguridad de AWS de establecer permisos limitados en las políticas de IAM.

Además de los permisos de Lambda, el usuario o rol que crea la función también debe tener los permisos `BatchGetImage` y `GetDownloadUrlForLayer`.

## Ciclo de vida de la función

Después de cargar una imagen de contenedor nueva o actualizada, Lambda la optimiza antes de que la función pueda procesar las invocaciones. El proceso de optimización puede tardar unos segundos. La función permanece en el estado `Pending` hasta que se completa el proceso. A continuación, la función pasa al estado `Active`. Mientras que el estado es `Pending`, puede invocar la función, pero otras operaciones en la función fallan. Las invocaciones que se producen mientras una actualización de imagen está en curso ejecutan el código de la imagen anterior.

Si una función no se invoca durante varias semanas, Lambda recupera su versión optimizada y la función pasa al estado `Inactive`. Para reactivar la función, debe invocarla. Lambda rechaza la primera invocación y la función entra en el estado `Pending` hasta que Lambda vuelva a optimizar la imagen. A continuación, la función regresa al `Active` estado.

Lambda recupera periódicamente la imagen del contenedor asociado del repositorio de Amazon ECR. Si la imagen de contenedor correspondiente ya no existe en Amazon ECR o se revocan los permisos, la función entra en estado `Failed` y Lambda devuelve un error para cualquier invocación de la función.

Puede utilizar la API de Lambda para obtener información sobre el estado de una función. Para obtener más información, consulte [Estados de función de Lambda](#).

# Configuración de la memoria de una función de Lambda

Lambda asigna potencia de CPU en proporción a la cantidad de memoria configurada. La memoria es la cantidad de memoria disponible para la función de Lambda en tiempo de ejecución. Puede aumentar o disminuir la memoria y la potencia de CPU asignada a su función mediante la configuración Memoria. Puede configurar un valor de memoria comprendido entre 128 MB y 10 240 MB, en incrementos de 1 MB. Si se configuran 1769 MB, la función tiene el equivalente de una vCPU (un segundo de créditos de vCPU por segundo).

En esta página se describe cómo y cuándo actualizar la configuración de memoria de una función de Lambda.

## Secciones

- [Determinación de la configuración de memoria adecuada de una función de Lambda](#)
- [Configuración de la memoria de función \(consola\)](#)
- [Configuración de la memoria de una función \(AWS CLI\)](#)
- [Configuración de la memoria de una función \(AWS SAM\)](#)
- [Aceptación de recomendaciones de memoria de función \(consola\)](#)

## Determinación de la configuración de memoria adecuada de una función de Lambda

La memoria es la palanca principal para controlar el rendimiento de una función. El valor predeterminado, 128 MB, es el más bajo posible. Solo se recomienda utilizar 128 MB para las funciones de Lambda sencillas, como las que transforman y enrutan eventos a otros servicios de AWS. Una mayor asignación de memoria puede mejorar el rendimiento de las funciones que utilizan bibliotecas importadas, [capas de Lambda](#), Amazon Simple Storage Service (Amazon S3) o Amazon Elastic File System (Amazon EFS). Agregar más memoria aumenta proporcionalmente la cantidad de CPU, lo que aumenta la potencia computacional general disponible. Si una función está vinculada a la CPU, la red o la memoria, aumentar la configuración de memoria puede mejorar drásticamente su rendimiento.

Para encontrar la configuración de memoria adecuada para sus funciones, recomendamos utilizar la herramienta de código abierto [Power Tuning de AWS Lambda](#). Esta herramienta utiliza AWS Step Functions para ejecutar varias versiones simultáneas de una función de Lambda en diferentes asignaciones de memoria y medir el rendimiento. La función de entrada se ejecuta en su cuenta de



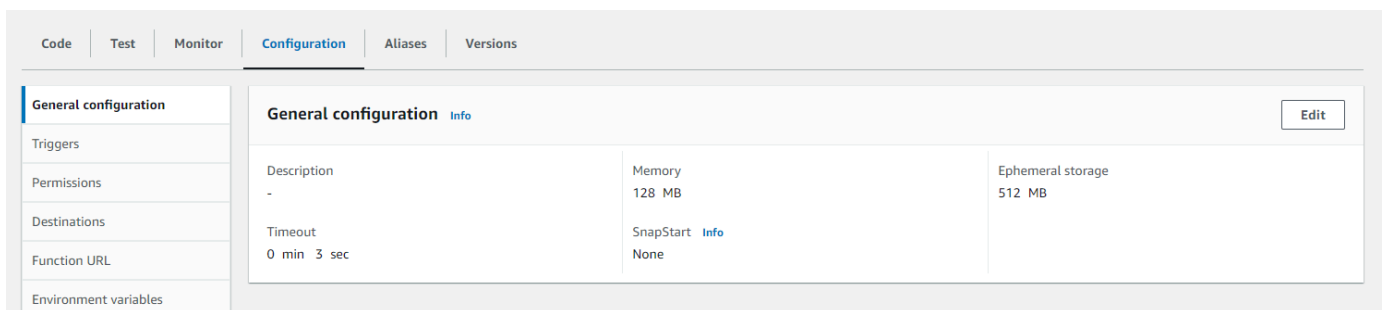
AWS y realiza llamadas HTTP en tiempo real e interacciones con el SDK para medir el rendimiento probable en un escenario de producción activo. También puede implementar un proceso de CI/CD para usar esta herramienta y medir automáticamente el rendimiento de las nuevas funciones que implemente.

## Configuración de la memoria de función (consola)

Puede configurar la memoria de su función en la consola de Lambda.

Para actualizar la memoria de una función

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Seleccione Configuración y, a continuación, Configuración general.



4. En Configuración general, seleccione la pestaña Etiquetas.
5. En Memoria, establezca un valor comprendido entre 128 MB y 10 240 MB.
6. Elija Guardar.

## Configuración de la memoria de una función (AWS CLI)

Puede usar el comando [update-function-configuration](#) para configurar la memoria de la función.

Example

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --memory-size 1024
```

## Configuración de la memoria de una función (AWS SAM)

Puede usar [AWS Serverless Application Model](#) para configurar la memoria de su función. Actualice la propiedad [MemorySize](#) de su archivo `template.yaml` y, a continuación, ejecute [sam deploy](#).

Example `template.yaml`

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 1024
      # Other function properties...
```

## Aceptación de recomendaciones de memoria de función (consola)

Si tiene permisos de administrador en AWS Identity and Access Management (IAM), puede optar por recibir recomendaciones de configuración de memoria de Lambda función de AWS Compute Optimizer. Para obtener instrucciones sobre cómo darse de alta en las recomendaciones de memoria para su cuenta u organización, consulte [Opción en su cuenta](#) en la Guía del usuario de AWS Compute Optimizer.

### Note

Compute Optimizer solo admite funciones que utilizan la arquitectura `x86_64`.

Luego de realizar la activación y de que su [función de Lambda cumpla con los requisitos de Compute Optimizer](#), puede ver y aceptar recomendaciones de memoria de la función desde Compute Optimizer de la consola de Lambda, en Configuración general.

# Configuración del almacenamiento efímero para funciones de Lambda

Lambda proporciona almacenamiento efímero para las funciones del directorio `/tmp`. Este almacenamiento es temporal y exclusivo de cada entorno de ejecución. Puede controlar la cantidad de almacenamiento efímero asignado a su función mediante la configuración Almacenamiento efímero. Puede configurar un valor de almacenamiento efímero comprendido entre 512 MB y 10 240 MB, en incrementos de 1 MB. Todos los datos almacenados en `/tmp` se cifran en reposo con una clave administrada por AWS.

En esta página, se describen los casos de uso más comunes y cómo actualizar el almacenamiento efímero de una función de Lambda.

## Secciones

- [Casos de uso habituales para aumentar el almacenamiento efímero](#)
- [Configuración del almacenamiento efímero \(consola\)](#)
- [Configuración del almacenamiento efímero \(AWS CLI\)](#)
- [Configuración del almacenamiento efímero \(AWS SAM\)](#)

## Casos de uso habituales para aumentar el almacenamiento efímero

Estos son algunos casos de uso comunes que se benefician del aumento del almacenamiento efímero:

- Trabajos de extracción, transformación y carga (ETL): aumente el almacenamiento efímero cuando el código realice cálculos intermedios o descargue otros recursos para completar el procesamiento. Si se aumenta el espacio temporal, se pueden ejecutar trabajos de ETL más complejos en las funciones de Lambda.
- Inferencia de machine learning (ML): muchas tareas de inferencia se basan en archivos de datos de referencia de gran tamaño, que incluyen bibliotecas y modelos. Con un almacenamiento efímero, puede descargar modelos de mayor tamaño de Amazon Simple Storage Service (Amazon S3) en `/tmp` y utilizarlos en su procesamiento.
- Procesamiento de datos: en el caso de las cargas de trabajo que descargan objetos de Amazon S3 en respuesta a eventos de S3, un mayor espacio de `/tmp` permite gestionar objetos de mayor tamaño sin necesidad de utilizar el procesamiento en memoria. Las cargas de

trabajo que crean archivos PDF o procesan contenido multimedia también se benefician de más almacenamiento efímero.

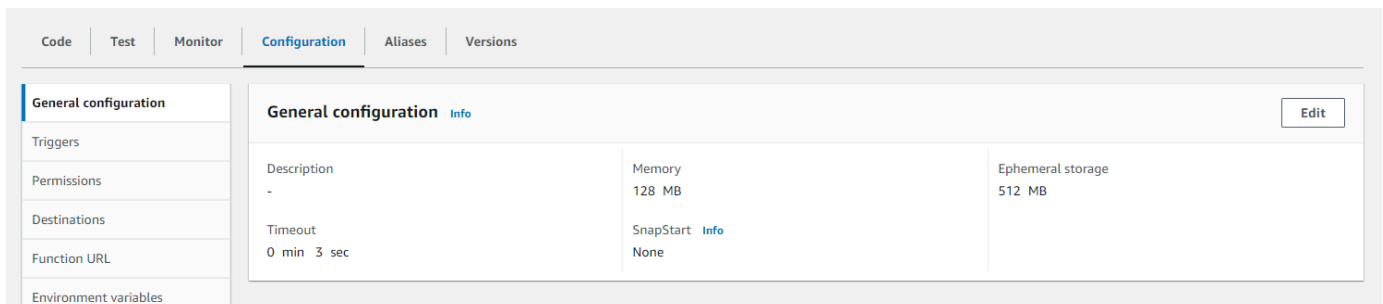
- **Procesamiento de gráficos:** el procesamiento de imágenes es un caso de uso habitual para las aplicaciones basadas en Lambda. Para las cargas de trabajo que procesan archivos TIFF o imágenes de satélite de gran tamaño, más almacenamiento efímero facilita el uso de bibliotecas y la realización de cálculos en Lambda.

## Configuración del almacenamiento efímero (consola)

Puede configurar el almacenamiento efímero en la consola de Lambda.

Modificación del almacenamiento efímero de una función

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Seleccione Configuración y, a continuación, Configuración general.



4. En Configuración general, seleccione la pestaña Etiquetas.
5. En Almacenamiento efímero, establezca un valor comprendido entre 512 MB y 10 240 MB, en incrementos de 1 MB.
6. Elija Guardar.

## Configuración del almacenamiento efímero (AWS CLI)

Puede usar el comando [update-function-configuration](#) para configurar el almacenamiento efímero.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
```

```
--ephemeral-storage '{"Size": 1024}'
```

## Configuración del almacenamiento efímero (AWS SAM)

Puede usar [AWS Serverless Application Model](#) para configurar el almacenamiento efímero de su función. Actualice la propiedad [EphemeralStorage](#) de su archivo `template.yaml` y, a continuación, ejecute [sam deploy](#).

### Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs20.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      # Other function properties...
```

# Configuración y selección de la arquitectura del conjunto de instrucciones para una función de Lambda

La arquitectura del conjunto de instrucciones de una función de Lambda determina el tipo de procesador informático que Lambda utiliza para ejecutar la función. Lambda proporciona una variedad de arquitecturas del conjunto de instrucciones:

- `arm64`: arquitectura ARM de 64 bits, para el procesador Graviton2 de AWS
- `x86_64`: arquitectura x86 de 64 bits, para procesadores basados en x86

## Note

La arquitectura `arm64` está disponible en la mayoría de Regiones de AWS. Para más información, consulte [Precios de AWS Lambda](#). En la tabla de precios de memoria, elija la pestaña ARM Price (Precio de ARM) y, a continuación, abra la lista desplegable Region (Región) para ver qué Regiones de AWS admiten `arm64` con Lambda.

Para obtener un ejemplo de cómo crear una función con arquitectura `arm64`, consulte [AWS Lambda Functions Powered by AWS Graviton2 Processor](#) (Funciones de con tecnología de Graviton2 Processor).

## Temas

- [Ventajas del uso de la arquitectura `arm64`](#)
- [Requisitos para migrar a la arquitectura `arm64`](#)
- [Compatibilidad de código de función con la arquitectura `arm64`](#)
- [Cómo migrar a la arquitectura `arm64`](#)
- [Configuración de la arquitectura del conjunto de instrucciones](#)

## Ventajas del uso de la arquitectura `arm64`

Las funciones de Lambda que utilizan arquitectura `arm64` (Procesador Graviton2 de AWS) pueden lograr un precio y un rendimiento significativamente mejores que la función equivalente que se ejecuta en la arquitectura `x86_64`. Considere el uso de `arm64` para aplicaciones de uso intensivo de informática, como la informática de alto rendimiento, la codificación de video y las cargas de trabajo de simulación.

La CPU Graviton2 utiliza el núcleo Neoverse N1 y admite Armv8.2 (incluidas las extensiones CRC y criptográficas) además de varias otras extensiones de arquitectura.

Graviton2 reduce el tiempo de lectura de la memoria mediante una caché L2 más grande por vCPU, lo que mejora el rendimiento de latencia de los backends web y móviles, los microservicios y los sistemas de procesamiento de datos. Graviton2 también proporciona un rendimiento de cifrado mejorado y admite conjuntos de instrucciones que mejoran la latencia de la inferencia de machine learning basada en CPU.

Para obtener más información acerca de Graviton2 de AWS, consulte [Procesador Graviton de AWS](#).

## Requisitos para migrar a la arquitectura arm64

Cuando selecciona una función de Lambda para migrar a la arquitectura arm64, para garantizar una migración fluida, asegúrese de que la función cumple los siguientes requisitos:

- El paquete de implementación contiene solo componentes de código abierto y código fuente que usted controla, para que pueda realizar las actualizaciones necesarias para la migración.
- Si el código de función incluye dependencias de terceros, cada biblioteca o paquete proporciona una versión de arm64.

## Compatibilidad de código de función con la arquitectura arm64

El código de función de Lambda debe ser compatible con la arquitectura del conjunto de instrucciones de la función. Antes de migrar una función a una arquitectura arm64, tenga en cuenta los siguientes aspectos sobre el código de función actual:

- Si agregó el código de función mediante el editor de código integrado, es probable que el código se ejecute en cualquiera de las arquitecturas sin modificaciones.
- Si cargó su código de función, debe cargar un código nuevo que sea compatible con la arquitectura de destino.
- Si la función utiliza capas, debe [verificar cada capa](#) para asegurarse de que sea compatible con la arquitectura nueva. Si una capa no es compatible, edite la función para reemplazar la versión de la capa actual por una versión de la capa compatible.
- Si la función utiliza extensiones de Lambda, debe verificar cada extensión para asegurarse de que sea compatible con la arquitectura nueva.

- Si la función utiliza un tipo de paquete de implementación de imágenes de contenedor, debe crear una imagen de contenedor nueva que sea compatible con la arquitectura de la función.

## Cómo migrar a la arquitectura arm64

Para migrar una función de Lambda a la arquitectura arm64, le recomendamos que siga los pasos a continuación:

1. Cree la lista de dependencias para su aplicación o carga de trabajo. Las dependencias frecuentes incluyen lo siguiente:
  - todas las bibliotecas y todos los paquetes que utiliza la función
  - las herramientas que utiliza para crear, implementar y probar la función, como los compiladores, los conjuntos de pruebas, las canalizaciones de integración continua y entrega continua (CI/CD), las herramientas de aprovisionamiento y los scripts
  - las extensiones de Lambda y las herramientas de terceros que utiliza para monitorear la función en producción
2. Para cada una de las dependencias, verifique la versión y, luego, verifique si hay versiones arm64 disponibles.
3. Cree un entorno para migrar su aplicación.
4. Arranque la aplicación.
5. Pruebe y depure la aplicación.
6. Pruebe el rendimiento de la función arm64. Compare el rendimiento con la versión x86\_64.
7. Actualice la canalización de su infraestructura para admitir las funciones de Lambda de arm64.
8. Organice la implementación en producción.

Por ejemplo, utilice la [configuración de enrutamiento de alias](#) para dividir el tráfico entre las versiones x86 y arm64 de la función y comparar el rendimiento y la latencia.

Para obtener más información acerca de cómo crear un entorno de código para la arquitectura arm64, incluida información específica del lenguaje para Java, Go, .NET y Python, consulte el repositorio de GitHub [Introducción a Graviton de AWS](#).



## Configuración de la arquitectura del conjunto de instrucciones

Puede configurar la arquitectura del conjunto de instrucciones para las funciones de Lambda nuevas y existentes mediante la consola de Lambda, los AWS SDK, la AWS Command Line Interface (AWS CLI) o AWS CloudFormation. Siga estos pasos para cambiar la arquitectura del conjunto de instrucciones para una función de Lambda existente desde la consola.

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función para la que desea configurar la arquitectura del conjunto de instrucciones.
3. En la pestaña Código principal, en la sección Configuración del tiempo de ejecución, seleccione Editar.
4. En Arquitectura, elija la arquitectura del conjunto de instrucciones que quiere que utilice la función.
5. Seleccione Guardar.

# Configuración del tiempo de espera de la función de Lambda

Lambda ejecuta el código durante un período de tiempo determinado antes de que se agote el tiempo de espera. El tiempo de espera es la cantidad máxima de tiempo en segundos que una función de Lambda puede ejecutarse. El valor predeterminado de esta configuración es de 3 segundos, pero puede ajustarlo en incrementos de 1 segundo hasta un valor máximo de 900 segundos (15 minutos).

En esta página se describe cómo y cuándo actualizar la configuración de tiempo de espera de una función de Lambda.

## Secciones

- [Determinación del valor de tiempo de espera adecuado de una función de Lambda](#)
- [Configuración del tiempo de espera \(consola\)](#)
- [Configuración del tiempo de espera \(AWS CLI\)](#)
- [Configuración del tiempo de espera \(AWS SAM\)](#)

## Determinación del valor de tiempo de espera adecuado de una función de Lambda

Si el valor del tiempo de espera se acerca a la duración media de una función, existe un mayor riesgo de que se agote el tiempo de espera de la función inesperadamente. La duración de una función puede variar en función de la cantidad de datos transferidos y procesados y de la latencia de cualquier servicio con el que interactúe la función. Algunas de las causas más comunes del tiempo de espera agotado son las siguientes:

- Las descargas desde Amazon Simple Storage Service (Amazon S3) son más grandes o tardan más que la media.
- Una función realiza una solicitud a otro servicio, lo que aumenta el tiempo de respuesta.
- Los parámetros proporcionados a una función requieren una mayor complejidad computacional en la función, lo que hace que la invocación tarde más tiempo.

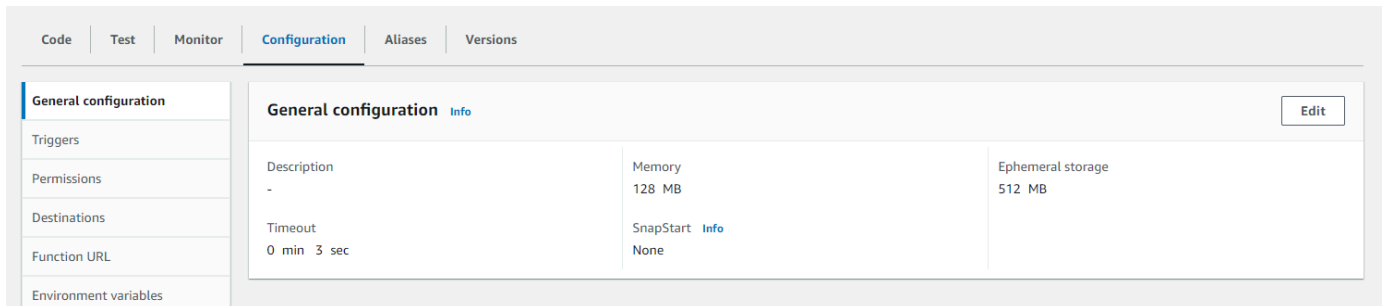
Al probar la aplicación, asegúrese de que las pruebas reflejen con precisión el tamaño y la cantidad de datos y de que los valores de los parámetros sean realistas. Las pruebas suelen utilizar muestras pequeñas por motivos de comodidad, pero debe utilizar conjuntos de datos que se sitúen en el límite superior de lo que cabe esperar razonablemente para su carga de trabajo.

## Configuración del tiempo de espera (consola)

Se puede configurar el tiempo de espera de la función en la consola de Lambda.

Para modificar el tiempo de espera de una función

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Seleccione Configuración y, a continuación, Configuración general.



4. En Configuración general, seleccione la pestaña Etiquetas.
5. En Tiempo de espera, establezca un valor comprendido entre 1 y 900 segundos (15 minutos).
6. Seleccione Guardar.

## Configuración del tiempo de espera (AWS CLI)

Puede usar el comando [update-function-configuration](#) para configurar el valor del tiempo de espera, en segundos. El siguiente comando de ejemplo aumenta el tiempo de espera de la función a 120 segundos (2 minutos).

Example

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --timeout 120
```

## Configuración del tiempo de espera (AWS SAM)

Puede usar [AWS Serverless Application Model](#) para configurar el valor de tiempo de espera de su función. Actualice la propiedad [Timeout](#) de su archivo `template.yaml` y, a continuación, ejecute [sam deploy](#).

## Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: An AWS Serverless Application Model template describing your function.  
Resources:  
  my-function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: .  
      Description: ''  
      MemorySize: 128  
      Timeout: 120  
      # Other function properties...
```

# Utilice variables de entorno Lambda para configurar valores en el código

Puede usar variables de entorno para ajustar el comportamiento de su función sin actualizar el código. Una variable de entorno es un par de cadenas almacenadas en la configuración específica de la versión de una función. El tiempo de ejecución de Lambda hace que las variables de entorno estén disponibles para el código y establece variables de entorno adicionales que contienen información sobre la función y la solicitud de invocación.

## Note

Para aumentar la seguridad, se recomienda utilizar AWS Secrets Manager en lugar de variables de entorno para almacenar las credenciales de la base de datos y otra información confidencial, como claves de API o tokens de autorización. Para obtener más información, consulte [Cree y administre secretos con AWS Secrets Manager](#).

Las variables de entorno no se evalúan antes de la invocación de la función. Cualquier valor que defina se considera una cadena literal y no expandida. Evalúe las variables en el código de la función.

Puede configurar las variables de entorno en Lambda mediante la consola de Lambda, la AWS Command Line Interface (AWS CLI), AWS Serverless Application Model (AWS SAM) o con un AWS SDK.

## Console

Las variables de entorno se definen en la versión no publicada de la función. Al publicar una versión, las variables de entorno se bloquean para esa versión junto con otra [configuración específica de la versión](#).

Puede crear una variable de entorno para su función al definir una clave y un valor. Su función utiliza el nombre de la clave para recuperar el valor de la variable de entorno.

Para establecer variables de entorno en la consola de Lambda

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuración y, a continuación, elija Variables de entorno.

4. En Variables de entorno, elija Editar.
5. Elija Add environment variable (Añadir variable de entorno).
6. Introduzca una clave y un valor.

#### Requisitos

- Las claves comienzan con una letra y tienen como mínimo dos caracteres.
- Las claves solo contienen letras, números y guiones bajos (\_).
- Las llaves no están [reservadas por Lambda](#).
- El tamaño total de todas las variables de entorno no supera los 4 KB.

7. Seleccione Guardar.

Para generar una lista de variables de entorno en el editor de código de la consola

Puede generar una lista de variables de entorno en el editor de código de Lambda. Esta es una forma rápida de hacer referencia a las variables de entorno mientras se codifica.

1. Elija la pestaña Código.
2. Seleccione la pestaña Variables de entorno.
3. Elija Herramientas, Mostrar variables de entorno.

Las variables de entorno permanecen cifradas cuando aparecen en el editor de código de la consola. Si habilitó los ayudantes de cifrado para el cifrado en tránsito, esa configuración permanecerá sin cambios. Para obtener más información, consulte [Asegurar las variables de entorno Lambda](#).

La lista de variables de entorno es de solo lectura y de uso exclusivo en la consola de Lambda. Este archivo no se incluye al descargar el archivo .zip de la función y no puede agregar variables de entorno al cargar este archivo.

#### AWS CLI

En el ejemplo siguiente se establecen dos variables de entorno en una función denominada `my-function`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --environment "Variables={BUCKET=amzn-s3-demo-bucket,KEY=file.txt}"
```

Cuando se aplican variables de entorno con el comando `update-function-configuration`, se reemplaza todo el contenido de la `Variables` estructura. Para conservar las variables de entorno existentes al agregar una nueva, incluya todos los valores existentes en la solicitud.

Para obtener la configuración actual, use el comando `get-function-configuration`.

```
aws lambda get-function-configuration \  
  --function-name my-function
```

Debería ver los siguientes datos de salida:

```
{  
  "FunctionName": "my-function",  
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",  
  "Runtime": "nodejs20.x",  
  "Role": "arn:aws:iam::111122223333:role/lambda-role",  
  "Environment": {  
    "Variables": {  
      "BUCKET": "amzn-s3-demo-bucket",  
      "KEY": "file.txt"  
    }  
  },  
  "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",  
  ...  
}
```

Puede pasar el ID de revisión de la salida de `get-function-configuration` como parámetro a `update-function-configuration`. Esto garantiza que los valores no cambien entre el momento en que lee la configuración y el momento en que la actualiza.

Para configurar la clave de cifrado de una función, establezca la opción `KMSKeyARN`.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --kms-key-arn arn:aws:kms:us-east-2:111122223333:key/055efbb4-xmpl-4336-  
ba9c-538c7d31f599
```

## AWS SAM

Puede utilizar [AWS Serverless Application Model](#) para configurar las variables de entorno de la función. Actualice las propiedades [Environment](#) y [Variables](#) del archivo `template.yaml` y, a continuación, ejecute [sam deploy](#).

## Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      Environment:
        Variables:
          BUCKET: amzn-s3-demo-bucket
          KEY: file.txt
      # Other function properties...
```

## AWS SDKs

Para administrar variables de entorno con un SDK de AWS, utilice las siguientes operaciones de la API.

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

Para obtener más información, consulte la [documentación del SDK de AWS](#) para el lenguaje de programación preferido.



## Variables definidas de entorno de tiempo de ejecución

Los [tiempos de ejecución](#) de Lambda establecen varias variables de entorno durante la inicialización. La mayoría de las variables de entorno proporcionan información sobre la función o el tiempo de ejecución. Las claves para estas variables de entorno están reservadas y no se pueden establecer en la configuración de la función.

### Variables de entorno reservadas

- `_HANDLER`: la localización del controlador configurada en la función.
- `_X_AMZN_TRACE_ID`: el [encabezado de rastreo de X-Ray](#). Esta variable de entorno cambia con cada invocación.
  - Esta variable de entorno no está definida para los tiempos de ejecución exclusivos del sistema operativo (la familia de tiempos de ejecución `provided`). Puede configurar `_X_AMZN_TRACE_ID` para tiempos de ejecución personalizados mediante el encabezado de respuesta `Lambda-Runtime-Trace-Id` de la [Siguiete invocación](#).
  - En las versiones 17 y posteriores de Java Runtime no se utiliza esta variable de entorno. En su lugar, Lambda almacena la información de rastreo en la propiedad del sistema `com.amazonaws.xray.traceHeader`.
- `AWS_DEFAULT_REGION`: la Región de AWS predeterminada donde se ejecuta la función de Lambda.
- `AWS_REGION`: la Región de AWS donde se ejecuta la función de Lambda. Si se define, este valor anula la `AWS_DEFAULT_REGION`.
  - Para obtener más información sobre cómo usar las variables de entorno de la Región de AWS con los SDK de AWS, consulte [Región de AWS](#) en la Guía de referencia de las herramientas y los SDK de AWS.
- `AWS_EXECUTION_ENV`: [identificador del tiempo de ejecución](#), precedido de `AWS_Lambda_` (por ejemplo, `AWS_Lambda_java8`). Esta variable de entorno no está definida para los tiempos de ejecución exclusivos del sistema operativo (la familia de tiempos de ejecución `provided`).
- `AWS_LAMBDA_FUNCTION_NAME`: el nombre de la función.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE`: la cantidad de memoria disponible para la función en MB.
- `AWS_LAMBDA_FUNCTION_VERSION`: la versión de la función que se está ejecutando.
- `AWS_LAMBDA_INITIALIZATION_TYPE`: el tipo de inicialización de la función, que es `on-demand`, `provisioned-concurrency` o `snap-start`. Para obtener información, consulte

## [Administración de la simultaneidad aprovisionada de Lambda](#) o [Mejora del rendimiento de inicio con Lambda SnapStart](#).

- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME`: el nombre del grupo de Registros de Amazon CloudWatch y flujo para la función. Las [variables de entorno](#) `AWS_LAMBDA_LOG_GROUP_NAME` y `AWS_LAMBDA_LOG_STREAM_NAME` no están disponibles en las funciones de Lambda SnapStart.
- `AWS_ACCESS_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`: las claves de acceso obtenidas del [rol de ejecución](#) de la función.
- `AWS_LAMBDA_RUNTIME_API`: ([Tiempo de ejecución personalizado](#)) El host y el puerto de la [API de tiempo de ejecución](#).
- `LAMBDA_TASK_ROOT`: la ruta al código de la función de Lambda.
- `LAMBDA_RUNTIME_DIR`: la ruta a las bibliotecas de tiempos de ejecución.

Las siguientes variables de entorno adicionales no están reservadas y pueden ampliarse en la configuración de la función.

### Variables de entorno sin reserva

- `LANG`: configuración regional del tiempo de ejecución (`en_US.UTF-8`).
- `PATH`: ruta de ejecución (`/usr/local/bin:/usr/bin:/bin:/opt/bin`).
- `LD_LIBRARY_PATH`: ruta de la biblioteca del sistema (`/var/lang/lib:/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- `NODE_PATH`: ([Node.js](#)) La ruta de la biblioteca Node.js (`/opt/nodejs/node12/node_modules:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `PYTHONPATH`: ([Python 2.7, 3.6, 3.8](#)) La ruta de la biblioteca de Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH`: ([Ruby](#)) La ruta de la biblioteca Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).
- `AWS_XRAY_CONTEXT_MISSING`: para el seguimiento de X-Ray, Lambda establece esto en `LOG_ERROR` para evitar arrojar errores de tiempo de ejecución desde el SDK de X-Ray.
- `AWS_XRAY_DAEMON_ADDRESS`: para el rastreo de X-Ray, la dirección IP y el puerto del daemon de X-Ray.
- `AWS_LAMBDA_DOTNET_PREJIT`: para los tiempos de ejecución de .NET 6 y .NET 7, establezca esta variable para habilitar o deshabilitar optimizaciones de tiempos de ejecución específicos

de .NET. Los valores incluyen `always`, `never`, y `provisioned-concurrency`. Para obtener más información, consulte [Configuración de simultaneidad aprovisionada para una función](#).

- TZ: la zona horaria del entorno (:UTC). El entorno de ejecución utiliza NTP para sincronizar el reloj del sistema.

Los valores de muestra presentados reflejan los últimos tiempos de ejecución. La presencia de variables específicas o sus valores pueden variar en tiempos de ejecución anteriores.

## Escenario de ejemplo para variables de entorno

Puede usar variables de entorno para personalizar el comportamiento de la función en su entorno de prueba y entorno de producción. Por ejemplo, puede crear dos funciones con el mismo código pero con configuración diferente. Una función se conecta a una base de datos de prueba y la otra se conecta a una base de datos de producción. En esta situación, utiliza variables de entorno para pasar el nombre de host y otros detalles de conexión de la base de datos a la función.

En el ejemplo siguiente se muestra cómo definir el host de base de datos y el nombre de base de datos como variables de entorno.

ENVIRONMENT	DEVELOPMENT	Remove
databaseHost	lambdadb	Remove
databaseName	rd1owwlydynnm5.cuovuayfg087	Remove
<i>Key</i>	<i>Value</i>	Remove

Si desea que su entorno de prueba genere más información de depuración que el entorno de producción, puede establecer una variable de entorno para configurar su entorno de prueba para utilizar un registro más detallado o un seguimiento más detallado.

## Asegurar las variables de entorno Lambda

A fin de proteger las variables de entorno, puede utilizar el cifrado en el servidor para proteger los datos en reposo y el cifrado del lado del cliente para proteger los datos en tránsito.

**Note**

Para aumentar la seguridad de la base de datos, se recomienda utilizar AWS Secrets Manager en lugar de variables de entorno para almacenar las credenciales de la base de datos. Para obtener más información, consulte [Uso de AWS Lambda con Amazon RDS](#).

## Seguridad en reposo

Lambda siempre proporciona cifrado en el servidor en reposo con una AWS KMS key. De forma predeterminada, Lambda utiliza una Clave administrada de AWS. Si este comportamiento predeterminado se ajusta a su flujo de trabajo, no tiene que configurar nada más. Lambda crea la Clave administrada de AWS en su cuenta y administra sus permisos. AWS no le cobrará por el uso de esta clave.

Si lo prefiere, puede proporcionar una clave administrada por el cliente de AWS KMS en su lugar. Puede hacerlo para tener control sobre la rotación de la clave de KMS o para cumplir con los requisitos de su organización para administrar claves de KMS. Cuando usa una clave administrada por el usuario, solo los usuarios de su cuenta con acceso a la clave de KMS pueden ver o administrar las variables de entorno de la función.

Las claves administradas por el cliente ocasionan cargos de AWS KMS estándar. Para más información, consulte [Precios de AWS Key Management Service](#).

## Seguridad en tránsito

Para mayor seguridad, puede habilitar las funciones auxiliares del cifrado en tránsito, de modo que garantiza que las variables de entorno se cifren en el lado del cliente para su protección en tránsito.

## Configuración del cifrado de las variables de entorno

1. Use AWS Key Management Service (AWS KMS) a fin de crear claves administradas por el cliente para que Lambda las utilice para el cifrado del lado del cliente y del servidor. Para obtener más información, consulte [Creación de claves](#) en la AWS Key Management Service Guía para desarrolladores de .
2. Mediante la consola de Lambda, vaya a la página Editar variables de entorno.
  - a. Abra la página de [Funciones](#) en la consola de Lambda.
  - b. Elija una función.

- c. Elija Configuración y, a continuación, elija Variables de entorno en la barra de navegación izquierda.
  - d. En la sección Variables de entorno, elija Editar.
  - e. Expanda Configuración de cifrado.
3. (Opcional) Habilite las funciones auxiliares de cifrado de la consola para que utilicen el cifrado del cliente a fin de proteger los datos en tránsito.
- a. En Cifrado en tránsito, elija Activar funciones auxiliares para el cifrado en tránsito.
  - b. Para cada variable de entorno para la que desee habilitar las funciones auxiliares de cifrado de la consola, elija Cifrar junto a la variable de entorno.
  - c. En la AWS KMS key para el cifrado en tránsito, elija una clave administrada por el cliente que haya creado al principio de este procedimiento.
  - d. Elija Política de rol de ejecución y copie la política. Esta política concede permiso al rol de ejecución de la función para descifrar las variables de entorno.  
  
Guarde esta política para usarla en el último paso de este procedimiento.
  - e. Agregue el código a la función que descifre las variables de entorno. Para ver un ejemplo, elija Fragmento para descifrar secretos.
4. (Opcional) Especifique su clave administrada por el cliente para el cifrado en reposo.
- a. Elija Use una clave maestra del cliente.
  - b. Elija una clave administrada por el cliente que haya creado al principio de este procedimiento.
5. Seleccione Guardar.
6. Configure los permisos.

Si está utilizando una clave administrada por el cliente con cifrado en el servidor, conceda permisos a cualquier usuario o rol que desee que pueda ver o administrar variables de entorno en la función. Para obtener más información, consulte [Administración de permisos para la clave de KMS de cifrado en el servidor](#).

Si habilita el cifrado del lado del cliente para la seguridad en tránsito, su función necesita permiso para llamar a la operación de la API kms : Decrypt. Agregue la política que guardó anteriormente en este procedimiento al [rol de ejecución](#) de la función.

## Administración de permisos para la clave de KMS de cifrado en el servidor

No se requieren permisos de AWS KMS para el usuario o el rol de ejecución de la función para utilizar la clave de cifrado predeterminada. Para utilizar una clave administrada por el cliente, necesita permisos de uso de la clave. Lambda usa sus permisos para crear una concesión en la clave. Esto permite a Lambda usarlo para el cifrado.

- `kms:ListAliases`: para ver las teclas en la consola de Lambda.
- `kms:CreateGrant`, `kms:Encrypt`: para configurar una clave administrada por el cliente en una función.
- `kms:Decrypt`: para ver y administrar variables de entorno cifradas con una clave administrada por el cliente.

Puede obtener estos permisos de su Cuenta de AWS o de la política de permisos basada en recursos de una clave. `ListAliases` surge de las [políticas administradas en Lambda](#). Las políticas clave conceden los permisos restantes a los usuarios del grupo Usuarios clave .

Los usuarios sin permisos `Decrypt` todavía pueden administrar funciones, pero no pueden ver variables de entorno ni administrarlas en la consola de Lambda. Para evitar que un usuario vea variables de entorno, añada una instrucción a los permisos del usuario que deniegue el acceso a la clave predeterminada, a una clave administrada por el cliente o a todas las claves.

Example Política de IAM: denegar acceso por ARN de clave

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Deny",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-2:111122223333:key/3be10e2d-xmpl-4be4-
bc9d-0405a71945cc"
    }
  ]
}
```

Para obtener información detallada sobre la administración de permisos de clave, consulte [Uso de políticas de claves en AWS KMS](#) en la Guía para desarrolladores de AWS Key Management Service.

## Recuperar variables de entorno Lambda

Para recuperar variables de entorno en el código de función, utilice el método estándar para el lenguaje de programación.

### Node.js

```
let region = process.env.AWS_REGION
```

### Python

```
import os
region = os.environ['AWS_REGION']
```

#### Note

En algunos casos, es posible que deba usar el siguiente formato:

```
region = os.environ.get('AWS_REGION')
```

### Ruby

```
region = ENV["AWS_REGION"]
```

### Java

```
String region = System.getenv("AWS_REGION");
```

### Go

```
var region = os.Getenv("AWS_REGION")
```

### C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

## PowerShell

```
$region = $env:AWS_REGION
```

Lambda almacena variables de entorno de forma segura cifrándolas en reposo. Puede [configurar Lambda para que utilice una clave de cifrado diferente](#), cifrar valores de variables de entorno del lado del cliente o establezca variables de entorno en una plantilla de AWS CloudFormation con AWS Secrets Manager.



# Otorgamiento a las funciones de Lambda de acceso a los recursos de una Amazon VPC

Con Amazon Virtual Private Cloud (Amazon VPC), usted puede crear redes privadas en su Cuenta de AWS para alojar recursos, como por ejemplo instancias de Amazon Elastic Compute Cloud (Amazon EC2), instancias de Amazon Relational Database Service (Amazon RDS) e instancias de Amazon ElastiCache. Puede conceder a su función de Lambda acceso a los recursos alojados en una Amazon VPC al adjuntar su función a la VPC a través de las subredes privadas que contienen los recursos. Siga las instrucciones de las siguientes secciones para adjuntar una función de Lambda a una Amazon VPC mediante la consola de Lambda, la AWS Command Line Interface (AWS CLI) o el AWS SAM.

## Note

Cada función de Lambda se ejecuta dentro de una VPC que pertenece y es administrada por el servicio Lambda. Lambda gestiona automáticamente estas VPC y no se encuentran visibles para los clientes. La configuración de la función para acceder a otros recursos de AWS en una Amazon VPC no afecta a la VPC gestionada por Lambda en la que se ejecuta la función.

## Secciones

- [Permisos de IAM necesarios](#)
- [Conexión de las funciones de Lambda a una Amazon VPC en su Cuenta de AWS](#)
- [Acceso a Internet cuando está conectado a una VPC](#)
- [Compatibilidad con IPv6](#)
- [Prácticas recomendadas para utilizar Lambda con Amazon VPC](#)
- [Introducción a las interfaces de red elástica \(ENI\) de hiperplano](#)
- [Uso de claves de condición de IAM para la configuración de la VPC](#)
- [Tutoriales de VPC](#)

## Permisos de IAM necesarios

Para adjuntar una función de Lambda a una Amazon VPC en su Cuenta de AWS, Lambda necesita permisos para crear y administrar las interfaces de red que utiliza para que su función acceda a los recursos de la VPC.

Las interfaces de red que Lambda crea se conocen como interfaces de red elásticas de hiperplano o ENI de hiperplano. Para obtener más información acerca de estas interfaces de red, consulte [the section called "Introducción a las interfaces de red elástica \(ENI\) de hiperplano"](#).

Puede conceder a la función los permisos que necesita adjuntando la [política administrada](#) de AWS AWSLambdaVPCAccessExecutionRole al rol de ejecución de la función. Cuando crea una función nueva en la consola de Lambda y la conecta a una VPC, Lambda añade automáticamente esta política de permisos.

Si prefiere crear su propia política de permisos de IAM, asegúrese de añadir los siguientes permisos:

- ec2:CreateNetworkInterface
- ec2:DescribeNetworkInterfaces: esta acción solo funciona si está permitida en todos los recursos ("Resource": "\*").
- ec2:DescribeSubnets
- ec2>DeleteNetworkInterface: si no especifica un ID de recurso para DeleteNetworkInterface en el rol de ejecución, es posible que la función no pueda acceder a la VPC. Especifique un ID de recurso único o incluya todos los ID de recursos, por ejemplo, "Resource": "arn:aws:ec2:us-west-2:123456789012:\*/\*".
- ec2:AssignPrivateIpAddresses
- ec2:UnassignPrivateIpAddresses

Tenga en cuenta que el rol de su función solo necesita estos permisos para crear las interfaces de red, no para invocar su función. Puede invocar correctamente la función cuando está conectada a una Amazon VPC, incluso si elimina estos permisos del rol de ejecución de la función.

Para adjuntar la función a una VPC, Lambda también necesita verificar los recursos de la red mediante su rol de usuario de IAM. Asegúrese de que su rol de usuario tenga los siguientes permisos de IAM:

- ec2:DescribeSecurityGroups

- ec2:DescribeSubnets
- ec2:DescribeVpcs

### Note

El servicio de Lambda utiliza los permisos de Amazon EC2 que concede al rol de ejecución de la función para adjuntar la función a una VPC. Sin embargo, estos permisos también se otorgan implícitamente al código de la función. Esto significa que su código de función puede realizar estas llamadas a la API de Amazon EC2. Para obtener consejos sobre prácticas recomendadas de seguridad, consulte [the section called “Prácticas recomendadas de seguridad”](#).

## Conexión de las funciones de Lambda a una Amazon VPC en su Cuenta de AWS

Adjunte su función a una Amazon VPC en su Cuenta de AWS mediante la consola de Lambda, la AWS CLI o el AWS SAM. Si utiliza AWS CLI o AWS SAM, o si adjunta una función existente a una VPC mediante la consola de Lambda, asegúrese de que el rol de ejecución de la función tenga los permisos necesarios que se indican en la sección anterior.


Las funciones Lambda no pueden conectarse directamente a una VPC con [tenencia de instancias dedicada](#). Para conectarse a los recursos de una VPC dedicada, [interconéctela con una segunda VPC con tenencia predeterminada](#).

### Lambda console

Cómo adjuntar una función a una Amazon VPC al crearla

1. Abra la página [Funciones](#) de la consola de Lambda y elija Crear función.
2. En Basic information (Información básica), para Function name (Nombre de función), escriba un nombre para la función.
3. Para configurar los ajustes de la VPC para la función, haga lo siguiente:
  - a. Amplíe Configuración avanzada.
  - b. Seleccione Habilitar VPC y, a continuación, elija la VPC a la que desea adjuntar la función.

- c. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
- d. Elija las subredes y los grupos de seguridad para crear la interfaz de red. Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.


 Note

Para acceder a recursos privados, conecte la función a subredes privadas. Si la función necesita acceso a Internet, consulte [the section called “Acceso a Internet para funciones de VPC”](#). La conexión de una función a una subred pública no le concede acceso a Internet ni una dirección IP pública.

4. Elija Crear función.

#### Cómo adjuntar una función existente a una Amazon VPC

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. Elija la pestaña Configuración y, a continuación, elija VPC.
3. Elija Editar.
4. En VPC, seleccione la Amazon VPC a la que quiera adjuntar su función.
5. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
6. Elija las subredes y los grupos de seguridad para crear la interfaz de red. Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.

 Note

Para acceder a recursos privados, conecte la función a subredes privadas. Si la función necesita acceso a Internet, consulte [the section called “Acceso a Internet para funciones de VPC”](#). La conexión de una función a una subred pública no le concede acceso a Internet ni una dirección IP pública.

7. Seleccione Guardar.

## AWS CLI

Cómo adjuntar una función a una Amazon VPC al crearla

- Para crear una función de Lambda y asociarla a una VPC, ejecute el siguiente comando de CLI `create-function`.

```
aws lambda create-function --function-name my-function \  
--runtime nodejs20.x --handler index.js --zip-file fileb://function.zip \  
--role arn:aws:iam::123456789012:role/lambda-role \  
--vpc-config  
  Ipv6AllowedForDualStack=true, SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036
```

Especifique sus propias subredes y grupos de seguridad y configure `Ipv6AllowedForDualStack` a `true` o `false` según su caso de uso.

Cómo adjuntar una función existente a una Amazon VPC

- Para asociar una función existente a una VPC, ejecute el siguiente comando de CLI `update-function-configuration`.

```
aws lambda update-function-configuration --function-name my-function \  
--vpc-config Ipv6AllowedForDualStack=true,  
  SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036, SecurityGroupIds=sg-0859123456789012
```

Cómo desvincular la función de una VPC

- Para desvincular la función de una VPC, ejecute el siguiente comando de CLI `update-function-configuration` con una lista vacía de subredes y grupos de seguridad de la VPC.

```
aws lambda update-function-configuration --function-name my-function \  
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

## AWS SAM

### Cómo adjuntar la función a una VPC

- Para adjuntar una función de Lambda a una VPC de Amazon, añada la propiedad `VpcConfig` a la definición de la función, tal y como se muestra en la siguiente plantilla de ejemplo. Para obtener más información sobre esta propiedad, consulte [AWS::Lambda::Function VPCConfig](#) en la Guía del usuario de AWS CloudFormation (la propiedad AWS SAM `VpcConfig` se pasa directamente a la propiedad `VpcConfig` de un recurso de AWS CloudFormation `AWS::Lambda::Function`).

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./lambda_function/
      Handler: lambda_function.handler
      Runtime: python3.12
      VpcConfig:
        SecurityGroupIds:
          - !Ref MySecurityGroup
        SubnetIds:
          - !Ref MySubnet1
          - !Ref MySubnet2
      Policies:
        - AWSLambdaVPCLambdaAccessExecutionRole

  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for Lambda function
      VpcId: !Ref MyVPC

  MySubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.1.0/24
```

```
MySubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyVPC
    CidrBlock: 10.0.2.0/24

MyVPC:
  Type: AWS::EC2::VPC
  Properties:
    CidrBlock: 10.0.0.0/16
```

Para obtener más información sobre la configuración de la VPC en AWS SAM, consulte [AWS::EC2::VPC](#) en la Guía del usuario de AWS CloudFormation.

## Acceso a Internet cuando está conectado a una VPC

De forma predeterminada, las funciones de Lambda tienen acceso al Internet público. Cuando asocie la función a una VPC, esta solo puede tener acceso a los recursos disponibles dentro de esa VPC. Para conceder a la función acceso a internet, también necesita configurar la VPC para tener acceso a Internet. Para obtener más información, consulte [the section called “Acceso a Internet para funciones de VPC”](#).

## Compatibilidad con IPv6

Su función puede conectarse a recursos en subredes de VPC de doble pila a través de IPv6. Esta opción está desactivada de forma predeterminada. Para permitir el tráfico IPv6 saliente, use la consola o la opción `--vpc-config Ipv6AllowedForDualStack=true` con el comando [create-function](#) o el comando [update-function-configuration](#).

### Note

Para permitir el tráfico IPv6 saliente en una VPC, todas las subredes que estén conectadas a la función deben ser subredes de doble pila. Lambda no admite conexiones IPv6 salientes para subredes exclusivas de IPv6 en una VPC, conexiones IPv6 salientes para funciones que no están conectadas a una VPC ni conexiones IPv6 entrantes mediante puntos de enlace de VPC (AWS PrivateLink).

Puede actualizar el código de función para conectarse explícitamente a los recursos de subred a través de IPv6. El siguiente ejemplo de Python abre un socket y se conecta a un servidor IPv6.

### Example — Conectarse al servidor IPv6

```
def connect_to_server(event, context):
    server_address = event['host']
    server_port = event['port']
    message = event['message']
    run_connect_to_server(server_address, server_port, message)

def run_connect_to_server(server_address, server_port, message):
    sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        # Send data
        sock.connect((server_address, int(server_port), 0, 0))
        sock.sendall(message.encode())
        BUFF_SIZE = 4096
        data = b''
        while True:
            segment = sock.recv(BUFF_SIZE)
            data += segment
            # Either 0 or end of data
            if len(segment) < BUFF_SIZE:
                break
        return data
    finally:
        sock.close()
```

## Prácticas recomendadas para utilizar Lambda con Amazon VPC

Para asegurarse de que la configuración de su Lambda VPC cumpla con las guías de las prácticas recomendadas, siga los consejos de las siguientes secciones.

### Prácticas recomendadas de seguridad

Para adjuntar la función de Lambda a una VPC, debe otorgar a su rol de ejecución de la función una cantidad de permisos de Amazon EC2. Estos permisos son necesarios para crear las interfaces de red que utiliza su función para acceder a los recursos de la VPC. Sin embargo, estos permisos también se otorgan implícitamente al código de la función. Esto significa que su código de función tiene permiso para realizar estas llamadas a la API de Amazon EC2.



Para seguir el principio del acceso con el privilegio mínimo, añade una política de denegación como la del siguiente ejemplo al rol de ejecución de la función. Esta política impide que su función realice llamadas a las API de Amazon EC2 que el servicio de Lambda utiliza para adjuntar su función a una VPC.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DetachNetworkInterface",
        "ec2:AssignPrivateIpAddresses",
        "ec2:UnassignPrivateIpAddresses",
      ],
      "Resource": [ "*" ],
      "Condition": {
        "ArnEquals": {
          "lambda:SourceFunctionArn": [
            "arn:aws:lambda:us-west-2:123456789012:function:my_function"
          ]
        }
      }
    }
  ]
}
```

AWS proporciona [grupos de seguridad](#) y [listas de control de acceso \(ACL\) de la red](#) para aumentar la seguridad en la VPC. Los grupos de seguridad controlan el tráfico de entrada y salida de los recursos, mientras que las ACL de red controlan el tráfico de entrada y salida de las subredes. Los grupos de seguridad proporcionan suficiente control de acceso para la mayoría de las subredes. Puede utilizar las ACL de red si desea agregar un nivel de seguridad adicional en la VPC. Para obtener instrucciones generales sobre las prácticas recomendadas de seguridad al utilizar Amazon VPC, consulte [Prácticas recomendadas de seguridad para su VPC](#) en la Guía del usuario de Amazon Virtual Private Cloud.

## Prácticas recomendadas de rendimiento

Al conectar la función a una VPC, Lambda comprueba si hay algún recurso de red disponible (ENI de hiperplano) al que pueda conectarse. Las ENI de hiperplano están asociadas a una combinación particular de grupos de seguridad y subredes de VPC. Si ya ha asociado una función a una VPC, especificando las mismas subredes y grupos de seguridad al adjuntar otra función, Lambda puede compartir los recursos de la red y evitar la necesidad de crear una nueva ENI de hiperplano. Para obtener más información sobre las ENI de hiperplano y su ciclo de vida, consulte [the section called “Introducción a las interfaces de red elástica \(ENI\) de hiperplano”](#).

## Introducción a las interfaces de red elástica (ENI) de hiperplano

Una ENI de hiperplano es un recurso gestionado que actúa como una interfaz de red entre la función de Lambda y los recursos a los que desea que se conecte la función. El servicio de Lambda crea y administra estas ENI automáticamente al conectar la función a una VPC.

Las ENI de hiperplano no son visibles directamente para usted y no necesita configurarlas ni administrarlas. Sin embargo, saber cómo funcionan puede ayudarle a comprender el comportamiento de su función cuando la adjunta a una VPC.

La primera vez que se asocia una función a una VPC mediante una combinación de subred y grupo de seguridad en particular, Lambda crea una ENI de hiperplano. Otras funciones de su cuenta que utilizan la misma combinación de subred y grupo de seguridad también pueden utilizar esta ENI. Siempre que es posible, Lambda reutiliza las ENI existentes para optimizar la utilización de los recursos y minimizar la creación de nuevas ENI. Sin embargo, cada ENI de hiperplano admite hasta 65 000 conexiones o puertos. Si el número de conexiones supera este límite, Lambda escala el número de ENI automáticamente en función del tráfico de red y los requisitos de simultaneidad.

En el caso de las funciones nuevas, mientras Lambda crea una ENI de hiperplano, la función permanece en el estado Pendiente y no se puede invocar. La función pasa al estado Activo solo cuando la ENI de hiperplano está lista, lo que puede tardar varios minutos. En el caso de las funciones existentes, no es posible realizar operaciones adicionales sobre la función, como crear nuevas versiones o actualizar su código, aunque aún se pueden invocar versiones anteriores de la misma.

### Note

Si una función de Lambda permanece inactiva durante 30 días, Lambda recupera las ENI de hiperplano no utilizadas y establece el estado de la función en inactivo. La siguiente

invocación fallará y la función reingresa al estado Pendiente hasta que Lambda completa la creación o asignación de una ENI de hiperplano. Para obtener más información acerca de los estados de función de Lambda, consulte [the section called “Estados de la función”](#).

## Uso de claves de condición de IAM para la configuración de la VPC

Puede utilizar claves de condición específicas de Lambda para la configuración de la VPC para proporcionar controles de permisos adicionales para sus funciones de Lambda. Por ejemplo, puede requerir que todas las funciones de la organización estén conectadas a una VPC. También puede especificar las subredes y los grupos de seguridad que los usuarios de la función pueden y no pueden utilizar.

Lambda admite las siguientes claves de condición en las políticas de IAM:

- `lambda:VpcIds`: permiten o deniegan una o varias VPC.
- `lambda:SubnetIds`: permiten o deniegan una o varias subredes.
- `lambda:SecurityGroupIds`: permiten o deniegan uno o varios grupos de seguridad.

Las operaciones de la API de Lambda [CreateFunction](#) y [UpdateFunctionConfiguration](#) admiten estas claves de condición. Para obtener más información acerca de las claves de condición en las políticas de IAM, consulte [Elementos de la política de JSON de IAM: Condición](#) en la Guía del usuario de IAM.

### Tip

Si su función ya incluye una configuración de VPC de una solicitud de la API anterior, puede enviar una solicitud `UpdateFunctionConfiguration` sin la configuración de la VPC.

## Políticas de ejemplo con claves de condición para la configuración de la VPC

En los ejemplos siguientes se muestra cómo utilizar claves de condición para la configuración de la VPC. Después de crear una instrucción de política con las restricciones deseadas, agregue la instrucción de política para el usuario o rol de destino.

Asegúrese de que los usuarios implementen solo funciones conectadas a la VPC

Para asegurarse de que todos los usuarios implementen solo funciones conectadas a la VPC, puede denegar operaciones de creación y actualización de funciones que no incluyan un ID de VPC válido.

Tenga en cuenta que el ID de VPC no es un parámetro de entrada para la solicitud `CreateFunction` o `UpdateFunctionConfiguration`. Lambda recupera el valor ID de la VPC en función de los parámetros de subred y grupo de seguridad.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
          "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Denegar a los usuarios el acceso a VPC, subredes o grupos de seguridad específicos

Para denegar a los usuarios el acceso a VPC específicas, utilice `StringEquals` para comprobar el valor de la condición `lambda:VpcIds`. En el ejemplo siguiente se deniega a los usuarios el acceso a `vpc-1` y `vpc-2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceOutOfVPC",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Effect": "Deny",
      "Resource": "*",

```

```

"Condition": {
  "StringEquals": {
    "lambda:VpcIds": ["vpc-1", "vpc-2"]
  }
}
}

```

Para denegar a los usuarios el acceso a subredes específicas, utilice `StringEquals` para comprobar el valor de la condición `lambda:SubnetIds`. En el ejemplo siguiente se deniega a los usuarios el acceso a `subnet-1` y `subnet-2`.

```

{
  "Sid": "EnforceOutOfSubnet",
  "Action": [
    "lambda:CreateFunction",
    "lambda:UpdateFunctionConfiguration"
  ],
  "Effect": "Deny",
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "lambda:SubnetIds": ["subnet-1", "subnet-2"]
    }
  }
}
}

```

Para denegar a los usuarios el acceso a grupos de seguridad específicos, utilice `StringEquals` para comprobar el valor de la condición `lambda:SecurityGroupIds`. En el ejemplo siguiente se deniega a los usuarios el acceso a `sg-1` y `sg-2`.

```

{
  "Sid": "EnforceOutOfSecurityGroups",
  "Action": [
    "lambda:CreateFunction",
    "lambda:UpdateFunctionConfiguration"
  ],
  "Effect": "Deny",
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {

```

```

        "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
    }
}
]
}

```

Permitir a los usuarios crear y actualizar funciones con configuraciones específicas de VPC

Para permitir a los usuarios acceder a VPC específicas, utilice `StringEquals` para comprobar el valor de la condición `lambda:VpcIds`. En el siguiente ejemplo se permite a los usuarios acceder a `vpc-1` y `vpc-2`.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceStayInSpecificVpc",
      "Action": [
        "lambda:CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "lambda:VpcIds": ["vpc-1", "vpc-2"]
        }
      }
    }
  ]
}

```

Para permitir a los usuarios acceder a subredes específicas, utilice `StringEquals` para comprobar el valor de la condición `lambda:SubnetIds`. En el siguiente ejemplo se permite a los usuarios acceder a `subnet-1` y `subnet-2`.

```

{
  "Sid": "EnforceStayInSpecificSubnets",
  "Action": [
    "lambda:CreateFunction",
    "lambda:UpdateFunctionConfiguration"
  ]
}

```

```
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "ForAllValues:StringEquals": {
        "lambda:SubnetIds": ["subnet-1", "subnet-2"]
      }
    }
  }
}
```

Para permitir a los usuarios acceder a grupos de seguridad específicos, utilice `StringEquals` para comprobar el valor de la condición `lambda:SecurityGroupIds`. En el siguiente ejemplo se permite a los usuarios acceder a `sg-1` y `sg-2`.

```
{
  "Sid": "EnforceStayInSpecificSecurityGroup",
  "Action": [
    "lambda:CreateFunction",
    "lambda:UpdateFunctionConfiguration"
  ],
  "Effect": "Allow",
  "Resource": "*",
  "Condition": {
    "ForAllValues:StringEquals": {
      "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
    }
  }
}
```

## Tutoriales de VPC

En los siguientes tutoriales, se conecta una función de Lambda a los recursos de la VPC.

- [Tutorial: Uso de una función de Lambda para obtener acceso a Amazon RDS en una Amazon VPC](#)
- [Tutorial: Configuración de una función de Lambda para obtener acceso a Amazon ElastiCache en una Amazon VPC](#)

# Habilitación del acceso a Internet para funciones de Lambda conectadas a VPC

De forma predeterminada, las funciones de Lambda se ejecutan en una VPC administrada por Lambda que tiene acceso a Internet. Para acceder a los recursos de una VPC de su cuenta, puede agregar una configuración de VPC a una función. Esto restringe la función a los recursos de esa VPC, a menos que la VPC tenga acceso a Internet. En esta página, se explica cómo proporcionar acceso a Internet a las funciones de Lambda conectadas por VPC.

## Aún no tengo una VPC

### Creación de la VPC

El Flujo de trabajo de creación de VPC crea todos los recursos de VPC necesarios para que una función de Lambda acceda a la Internet pública desde una subred privada, incluidas las subredes, la puerta de enlace NAT, la puerta de enlace de Internet y las entradas de la tabla de enrutamiento.

### Para crear la VPC

1. Abra la consola de Amazon VPC en <https://console.aws.amazon.com/vpc/>.
2. En el panel, elija Crear VPC.
3. En Recursos para crear, elija VPC y más.
4. Configurar la VPC
  - a. En Generación automática de etiquetas de nombre, ingrese un nombre para la VPC.
  - b. En Bloque de CIDR IPv4, puede conservar la sugerencia predeterminada o, como alternativa, ingresar el bloque de CIDR necesario para su aplicación o red.
  - c. Si la aplicación se comunica mediante direcciones IPv6, elija Bloque de CIDR IPv6, Bloque de CIDR IPv6 proporcionado por Amazon.
5. Configurar las subredes
  - a. Para Número de zonas de disponibilidad, elija 2. Se recomiendan como mínimo dos AZ para una alta disponibilidad.
  - b. Para Número de subredes públicas, elija 2.
  - c. Para Número de subredes privadas, elija 2.



- d. Puede conservar el bloque de CIDR predeterminado para la subred pública o, si lo prefiere, puede ampliar Personalizar los bloques de CIDR de la subred e introducir un bloque de CIDR. Para obtener más información, consulte [Subred de bloques CIDR](#).
6. En Puertas de enlace NAT, elija 1 por AZ para mejorar la resiliencia.
7. Para Puerta de enlace de Internet de solo salida, elija Sí si optó por incluir un bloque CIDR de IPv6.
8. En Puntos de conexión de VPC, mantenga el valor predeterminado (Puerta de enlace de S3). Esta opción no tiene ningún costo. Para obtener más información, consulte [Tipos de puntos de conexión de VPC de Amazon S3](#).
9. Mantenga la configuración predeterminada para las opciones DNS.
10. Seleccione Crear VPC.

## Configuración de la función de Lambda

### Para configurar una VPC al crear una función

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija Crear función.
3. En Basic information (Información básica), para Function name (Nombre de función), escriba un nombre para la función.
4. Amplíe Configuración avanzada.
5. Seleccione Habilitar VPC y, a continuación, elija una VPC.
6. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
7. En Subredes, seleccione todas las subredes privadas. Las subredes privadas pueden obtener acceso a Internet a través de una puerta de enlace NAT. La conexión de una función a una subred pública no le concede acceso a Internet.

#### Note

Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.

8. En el caso de los Grupos de seguridad, seleccione un grupo de seguridad que permita el tráfico saliente.

## 9. Elija Crear función.

Lambda crea automáticamente un rol de ejecución con la política [AWSLambdaVPCAccessExecutionRole](#) administrada por AWS. Los permisos de esta política solo son necesarios para crear interfaces de redes elásticas para la configuración de la VPC, no para invocar la función. Para aplicar permisos con privilegios mínimos, puede eliminar la política [AWSLambdaVPCAccessExecutionRole](#) de su rol de ejecución después de crear la función y la configuración de VPC. Para obtener más información, consulte [Permisos de IAM necesarios](#).

Para configurar una VPC para una función existente

Para agregar una configuración de VPC a una función existente, el rol de ejecución de la función debe tener [permiso para crear y administrar interfaces de redes elásticas](#). La política [AWSLambdaVPCAccessExecutionRole](#) administrada por AWS incluye los permisos requeridos. Para aplicar permisos con privilegios mínimos, puede eliminar la política [AWSLambdaVPCAccessExecutionRole](#) de su rol de ejecución después de crear la configuración de VPC.

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Configuración y, a continuación, elija VPC.
4. En VPC, elija Edit (Editar).
5. Seleccione la VPC.
6. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
7. En Subredes, seleccione todas las subredes privadas. Las subredes privadas pueden obtener acceso a Internet a través de una puerta de enlace NAT. La conexión de una función a una subred pública no le concede acceso a Internet.

### Note

Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.

8. En el caso de los Grupos de seguridad, seleccione un grupo de seguridad que permita el tráfico saliente.

## 9. Seleccione Guardar.

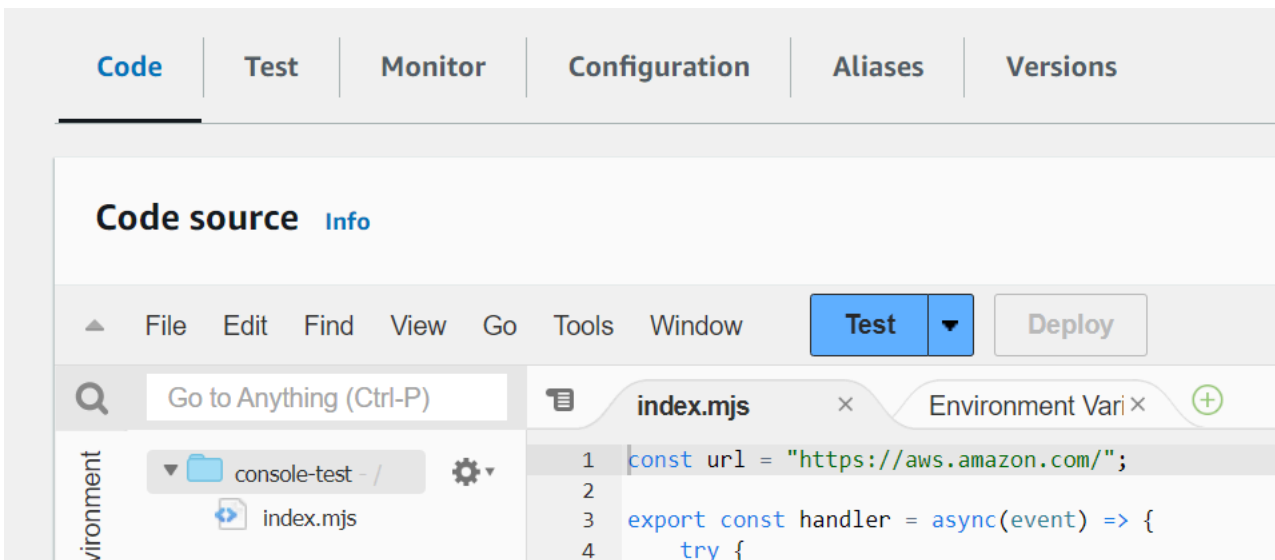
### Prueba de la función

Use el siguiente código de ejemplo para confirmar que su función conectada a VPC puede llegar a la Internet pública. Si se ejecuta correctamente, el código devuelve un código de estado 200. Si no funciona, la función agota el tiempo de espera.

### Node.js

En este ejemplo, se utiliza `fetch`, que está disponible en el tiempo de ejecución de `nodejs18.x` y posteriores.

1. En el panel Código fuente de la consola de Lambda, pegue el siguiente código en el archivo `index.mjs`. La función realiza una solicitud HTTP GET a un punto de conexión público y devuelve el código de respuesta HTTP para comprobar si la función tiene acceso a la Internet pública.



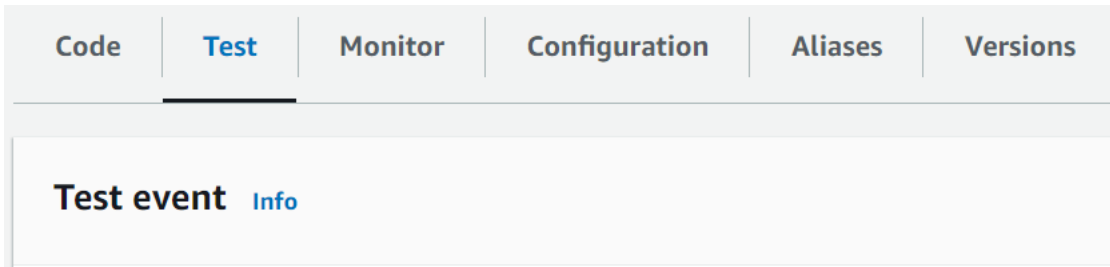
### Example — Solicitud HTTP con `async/await`

```
const url = "https://aws.amazon.com/";

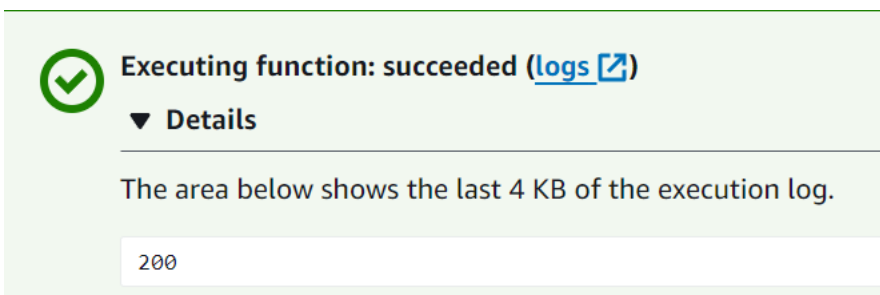
export const handler = async(event) => {
  try {
    // fetch is available with Node.js 18 and later runtimes
    const res = await fetch(url);
    console.info("status", res.status);
```

```
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

2. Elija Implementar.
3. Elija la pestaña Prueba.



4. Seleccione Probar.
5. La función devuelve un código de estado 200. Esto significa que la función tiene acceso saliente a Internet.

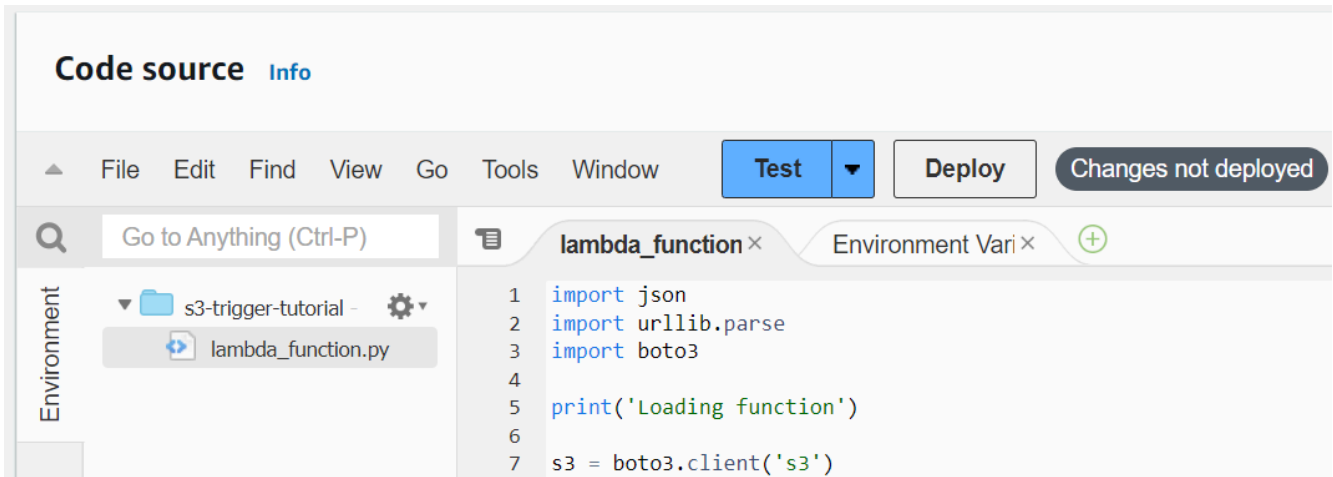


Si la función no puede acceder a la Internet pública, aparecerá un mensaje de error como este:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
  Task timed out after 3.01 seconds"
}
```

## Python

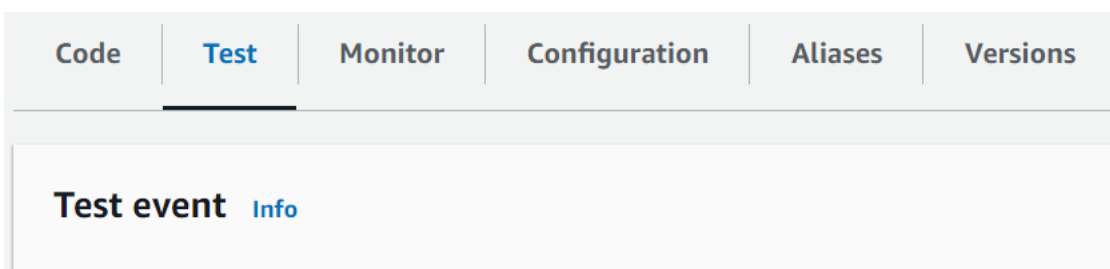
1. En el panel Código fuente de la consola de Lambda, pegue el siguiente código en el archivo `lambda_function.py`. La función realiza una solicitud HTTP GET a un punto de conexión público y devuelve el código de respuesta HTTP para comprobar si la función tiene acceso a la Internet pública.



```
import urllib.request

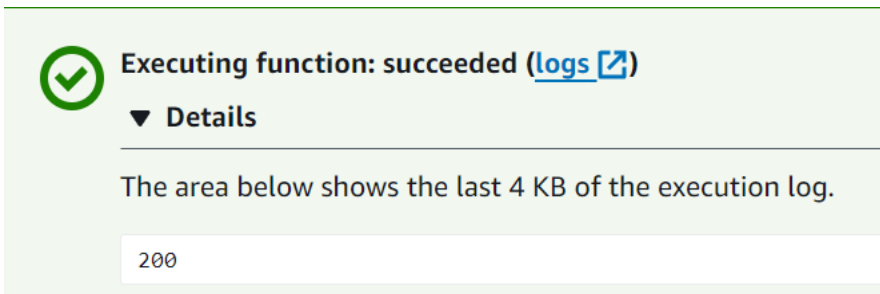
def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

2. Elija Implementar.
3. Elija la pestaña Prueba.



4. Seleccione Probar.

- La función devuelve un código de estado 200. Esto significa que la función tiene acceso saliente a Internet.



Si la función no puede acceder a la Internet pública, aparecerá un mensaje de error como este:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

## Ya tengo una VPC

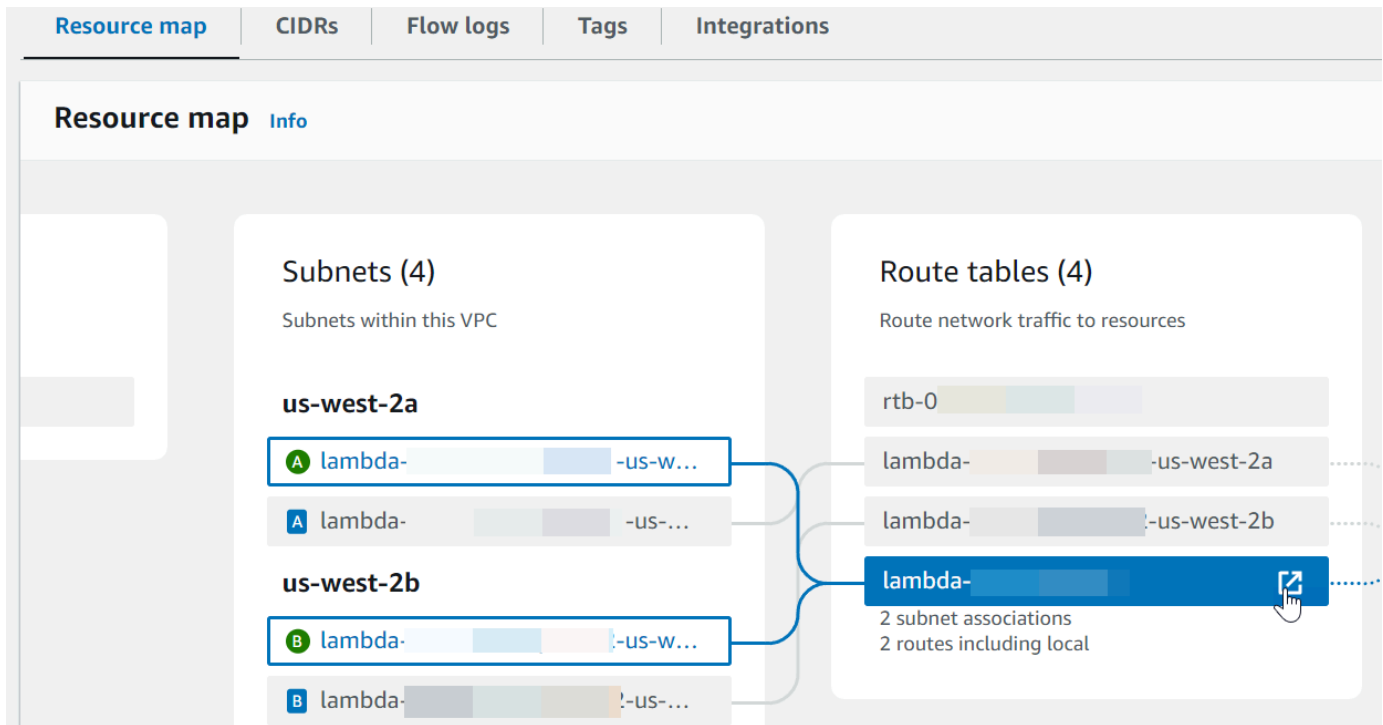
Si ya tiene una VPC, pero necesita configurar el acceso público a Internet para una función de Lambda, siga estos pasos. Este procedimiento supone que la VPC tiene, como mínimo, dos subredes. Si no tiene dos subredes, consulte [Crear una subred](#) en la Guía del usuario de Amazon VPC.

Verifique la configuración de la tabla de enrutamiento

- Abra la consola de Amazon VPC en <https://console.aws.amazon.com/vpc/>.
- Elija el ID de la VPC.

Your VPCs (3) <a href="#">Info</a>			
<input type="text" value="Search"/>			
<input type="checkbox"/>	Name	VPC ID	State
<input type="checkbox"/>	-	vpc-2[redacted]	Available
<input type="checkbox"/>	lambda-test-vpc	vpc-0[redacted]	Available

- Desplácese hasta la sección Mapa de recursos. Tenga en cuenta las asignaciones de tablas de enrutamiento. Abra cada tabla de enrutamiento que esté mapeada a una subred.



- Desplácese hasta la pestaña Rutas. Revise las rutas para determinar si se cumple una de las siguientes opciones. Cada uno de estos requisitos debe cumplirse mediante una tabla de enrutamiento independiente.
  - El tráfico con destino a Internet ( $0.0.0.0/0$  para IPv4,  $::/0$  para IPv6) se enruta a una puerta de enlace de Internet (`igw-xxxxxxxxxx`). Esto significa que la subred asociada a la tabla de enrutamiento es una subred pública.

**Note**

Si su subred no tiene un bloque CIDR de IPv6, solo verá la ruta IPv4 ( $0.0.0.0/0$ ).

## Example tabla de enrutamiento de la subred pública

Routes	Subnet associations	Edge associations	Route propagation	Tags
<b>Routes (4)</b>				
<input type="text" value="Filter routes"/>				
Destination	Target	Status		
::/0	<a href="#">igw-0</a>	Active		
::/56	local	Active		
0.0.0.0/0	<a href="#">igw-0</a>	Active		
/16	local	Active		

- El tráfico con destino a Internet para IPv4 (0.0.0.0/0) se enruta a una puerta de enlace NAT (nat-xxxxxxxxxx) que está asociada a una subred pública. Esto significa que la subred es privada y puede obtener acceso a Internet a través de una puerta de enlace NAT.

### Note

Si la subred tiene un bloque de CIDR IPv6, la tabla de enrutamiento también debe direccionar el tráfico IPv6 entrante (:::/0) a una puerta de enlace de Internet de solo salida (eigw-xxxxxxxxxx). Si su subred no tiene un bloque CIDR de IPv6, solo verá la ruta IPv4 (0.0.0.0/0).



## Example tabla de enrutamiento de subred privada

Routes	Subnet associations	Edge associations	Route propagation	Tags
<b>Routes (4)</b>				
<input type="text" value="Filter routes"/>				
Destination	Target	Status		
::/0	<a href="#">eigw-0</a>	Active		
::/56	local	Active		
0.0.0.0/0	<a href="#">nat-0</a>	Active		
/16	local	Active		

- Repita el paso anterior hasta que haya revisado cada tabla de enrutamiento asociada a una subred en su VPC y haya confirmado que tiene una tabla de enrutamiento con una puerta de enlace de Internet y una tabla de enrutamiento con una puerta de enlace NAT.

Si no tiene dos tablas de enrutamiento, una con una ruta a una puerta de enlace de Internet y otra con una ruta a una puerta de enlace NAT, siga estos pasos para crear los recursos y las entradas de la tabla de enrutamiento que faltan.

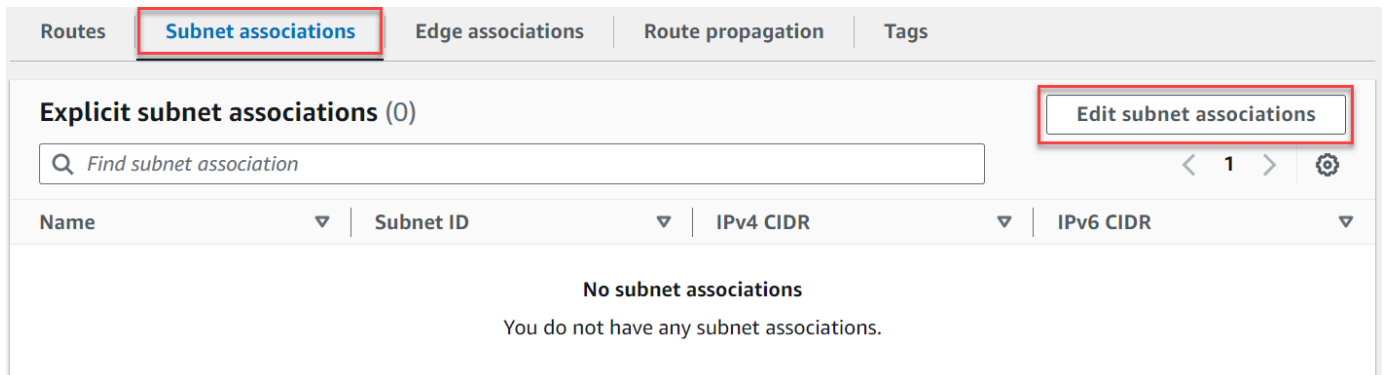
### Crear una tabla de enrutamiento

Siga estos pasos para crear una tabla de enrutamiento y asociarla a una subred.

#### Creación de una tabla de enrutamiento personalizada mediante la consola de Amazon VPC

- Abra la consola de Amazon VPC en <https://console.aws.amazon.com/vpc/>.
- En el panel de navegación, elija Tablas de enrutamiento.
- Elija Create Route Table (Crear tabla de enrutamiento).
- (Opcional) En Name (Etiqueta), escriba el nombre de la tabla de enrutamiento.
- En VPC, elija su VPC.
- (Opcional) Para agregar una etiqueta, elija Add new tag (Agregar etiqueta nueva) e ingrese la clave y el valor de la etiqueta.

7. Elija Create Route Table (Crear tabla de enrutamiento).
8. En la pestaña Subnet associations (Asociaciones de subred), elija Edit subnet associations (Editar asociaciones de subred).



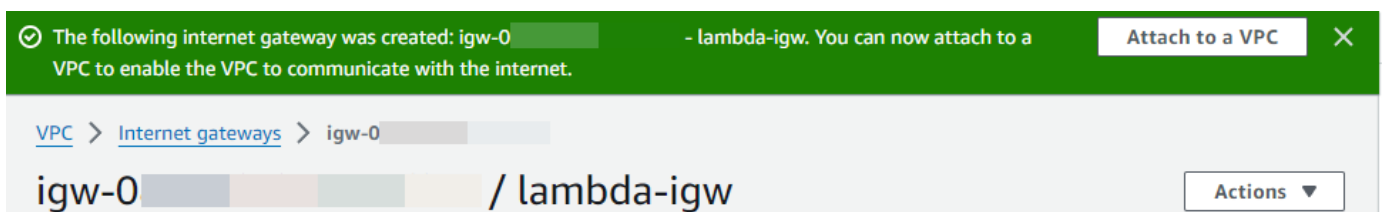
9. Seleccione la casilla de verificación para la subred que desee asociar a la tabla de enrutamiento.
10. Seleccione Save associations (Guardar asociaciones).

Cree un puerto de enlace de Internet

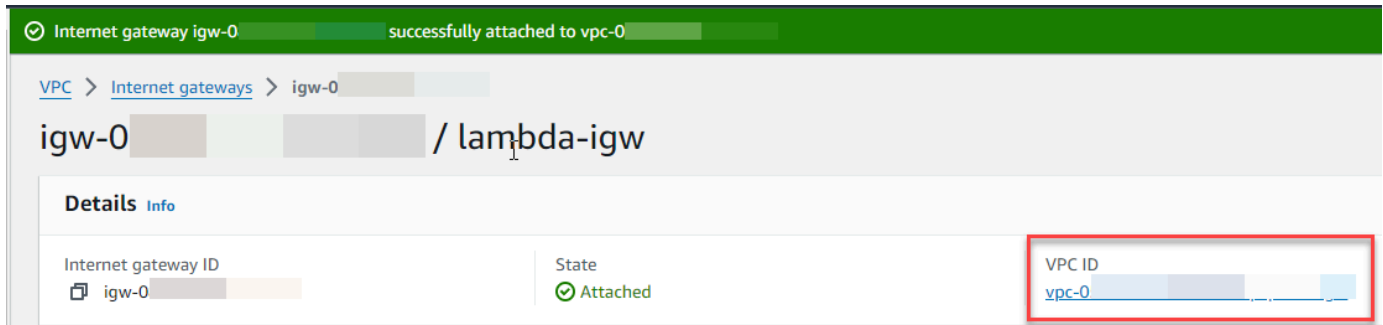
Siga estos pasos para crear una puerta de enlace de Internet, adjuntarla a su VPC y agregarla a la tabla de enrutamiento de la subred pública.

Para crear una puerta de enlace de Internet

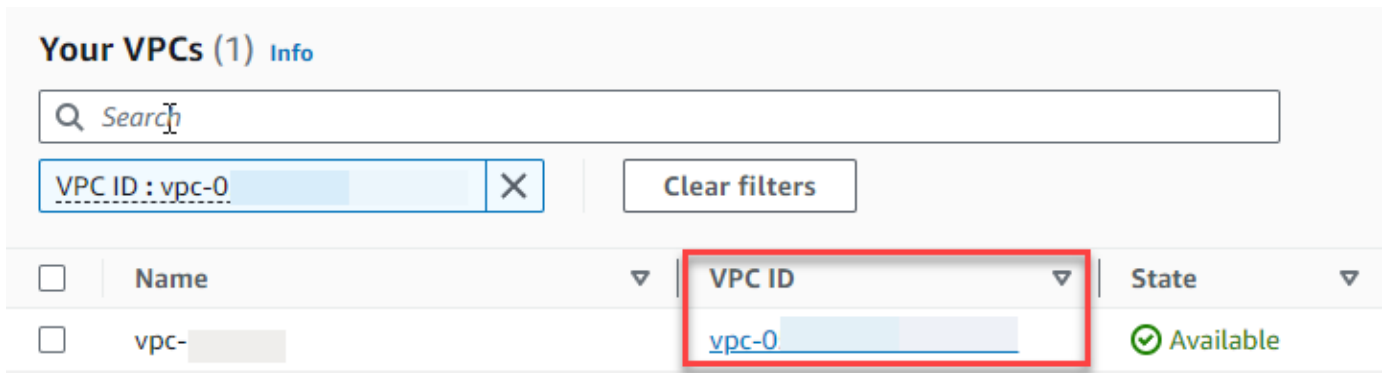
1. Abra la consola de Amazon VPC en <https://console.aws.amazon.com/vpc/>.
2. En el panel de navegación, elija Internet Gateways (Puertas de enlace de Internet).
3. Elija Crear puerta de enlace de Internet.
4. (Opcional) Ingrese un nombre para la puerta de enlace de Internet.
5. (Opcional) Para agregar una etiqueta, elija Agregar etiqueta nueva e ingrese la clave y el valor de la etiqueta.
6. Elija Crear puerta de enlace de Internet.
7. Elija Asociar a una VPC en el banner de la parte superior de la pantalla, seleccione una VPC disponible y, a continuación, elija Conectar puerta de enlace de Internet.



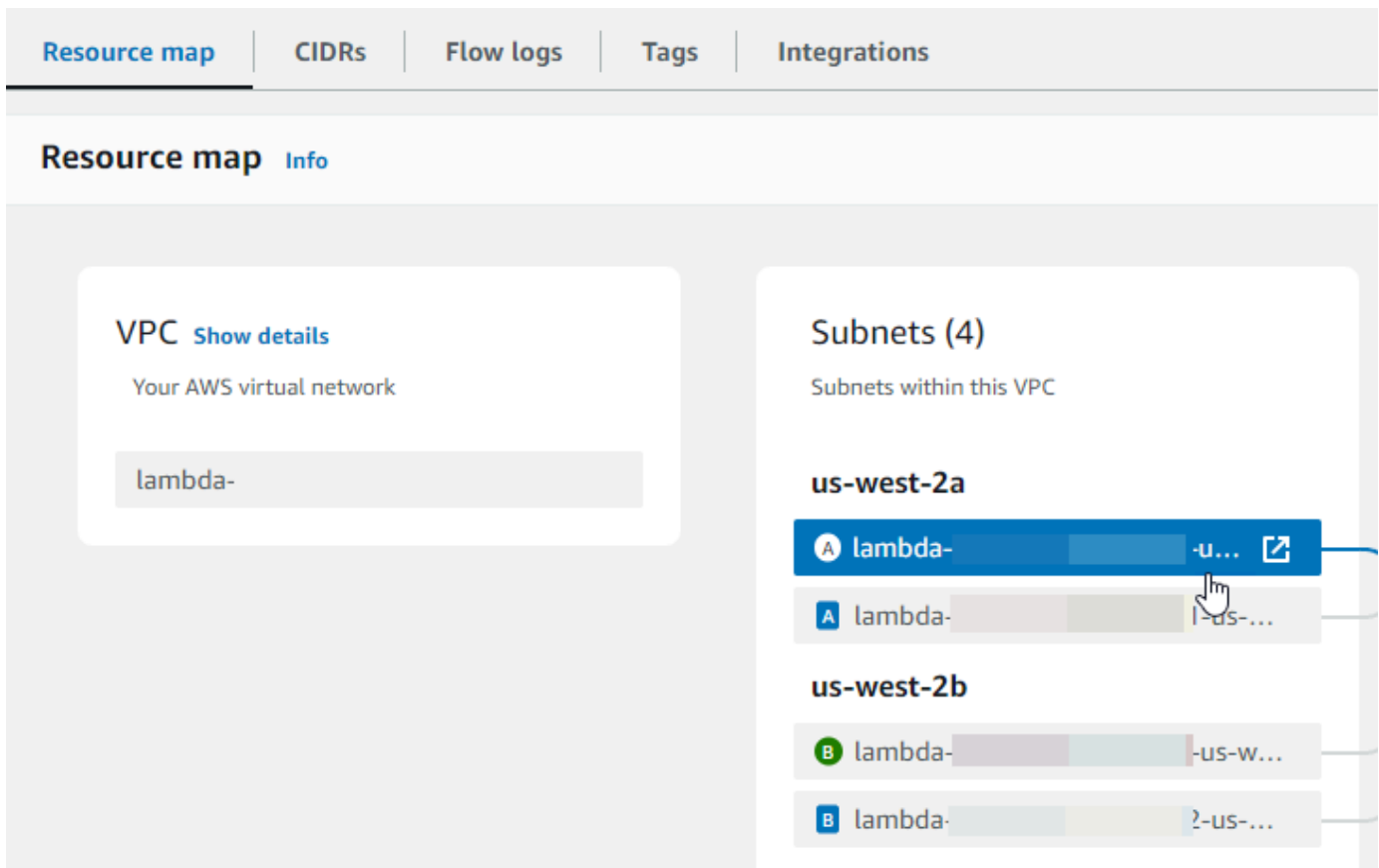
## 8. Elija el ID de la VPC.



## 9. Elija el ID de la VPC para abrir la página de detalles de VPC.

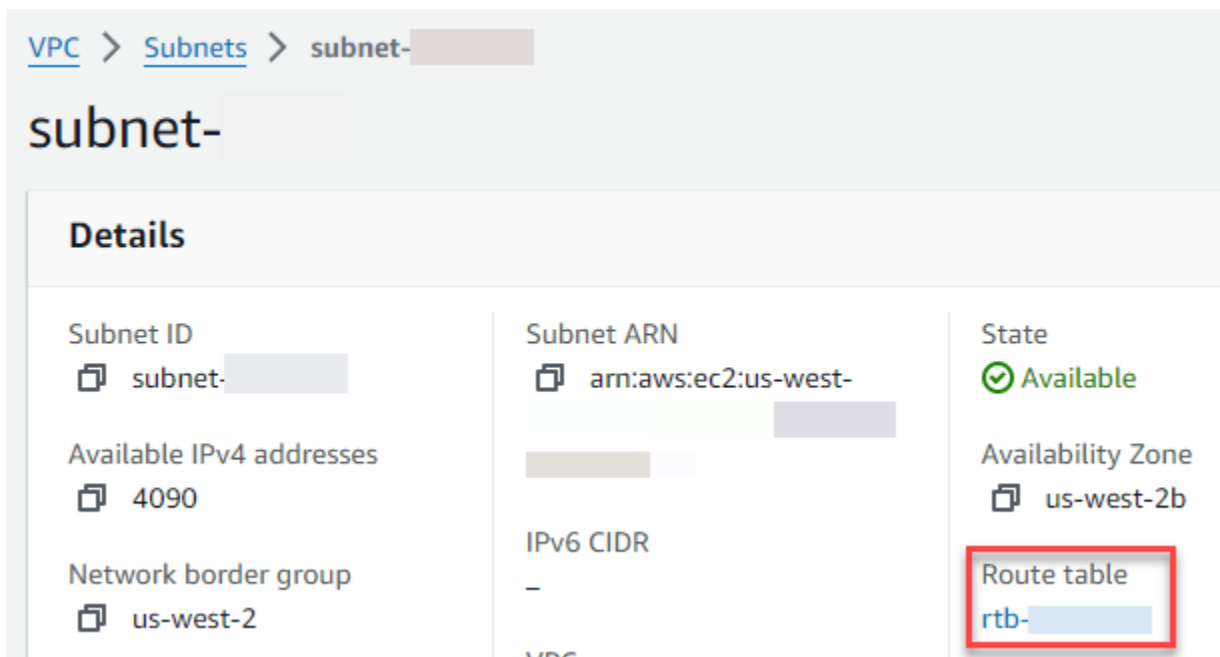


## 10. Desplácese hacia abajo hasta la sección Mapa de recursos y, a continuación, elija una subred. Los detalles de la subred se mostrarán en una nueva pestaña.



The screenshot shows the AWS Resource Map interface. At the top, there are navigation tabs: Resource map (selected), CIDRs, Flow logs, Tags, and Integrations. Below the tabs, the 'Resource map' section is active, with an 'Info' link. On the left, a 'VPC' card is visible with a 'Show details' link and the text 'Your AWS virtual network'. Below this, a search bar contains the text 'lambda-'. On the right, a 'Subnets (4)' card is shown, listing subnets within the VPC. The subnets are grouped by availability zone: 'us-west-2a' and 'us-west-2b'. Under 'us-west-2a', there are two subnets, the first of which is highlighted in blue and has a mouse cursor pointing to its 'Route table' link. Under 'us-west-2b', there are two subnets, the first of which is highlighted in green.

11. Elija el enlace en Tabla de enrutamiento.



The screenshot shows the AWS Subnet details page. The breadcrumb navigation is 'VPC > Subnets > subnet-'. The main heading is 'subnet-'. Below this, the 'Details' section is displayed. It contains several key-value pairs: 'Subnet ID' (subnet-), 'Subnet ARN' (arn:aws:ec2:us-west-), 'State' (Available), 'Available IPv4 addresses' (4090), 'Network border group' (us-west-2), 'IPv6 CIDR' (-), and 'Route table' (rtb-). The 'Route table' value is highlighted with a red box.

12. Elija el ID de la tabla de enrutamiento para abrir la página de detalles de la tabla de enrutamiento.

**Route tables (1) Info**

Find resources by attribute or tag

Route table ID : rtb-0 X Clear filters

<input type="checkbox"/>	Name	Route table ID
<input type="checkbox"/>	-	rtb-0

13. En Rutas, elija Editar rutas.

**Routes (1)** Both Edit routes

Filter routes

Destination	Target	Status
10.0.0.0/24	local	Active

14. Seleccione Agregar ruta y, a continuación, introduzca  $0.0.0.0/0$  en el cuadro Destino.

**Edit routes**

Destination	Target	Status
10.0.0.0/24	local	Active
0.0.0.0/0	local	-
0.0.0.0/8	-	-
0.0.0.0/16	-	-

15. En Objetivo, seleccione Puerta de enlace de Internet y, a continuación, elija la puerta de enlace de Internet que creó anteriormente. Si la subred tiene un bloque de CIDR IPv6, también debe agregar una ruta  $::/0$  para la misma puerta de enlace de Internet.

## Edit routes

Destination	Target
10.0.0.0/24	local
<input type="text" value="0.0.0.0/0"/>	<input type="text" value="local"/>
<input type="button" value="Add route"/>	<ul style="list-style-type: none"> <li>Carrier Gateway</li> <li>Core Network</li> <li>Egress Only Internet Gateway</li> <li>Gateway Load Balancer Endpoint</li> <li>Instance</li> <li><b>Internet Gateway</b></li> </ul>

16. Elija Guardar cambios.

### Creación de una gateway NAT

Siga estos pasos para crear una puerta de enlace NAT, asociarla a una subred pública y, a continuación, agregarla a la tabla de enrutamiento de la subred privada.

### Creación de una puerta de enlace NAT y asociación a una subred pública

1. En el panel de navegación, elija Puertas de enlace de NAT.
2. Elija Crear una puerta de enlace de NAT.
3. (Opcional) Ingrese un nombre para la puerta de enlace NAT.
4. En Subred, seleccione la subred pública de su VPC. (Una subred pública es una subred que tiene una ruta directa a una puerta de enlace de Internet en su tabla de enrutamiento).

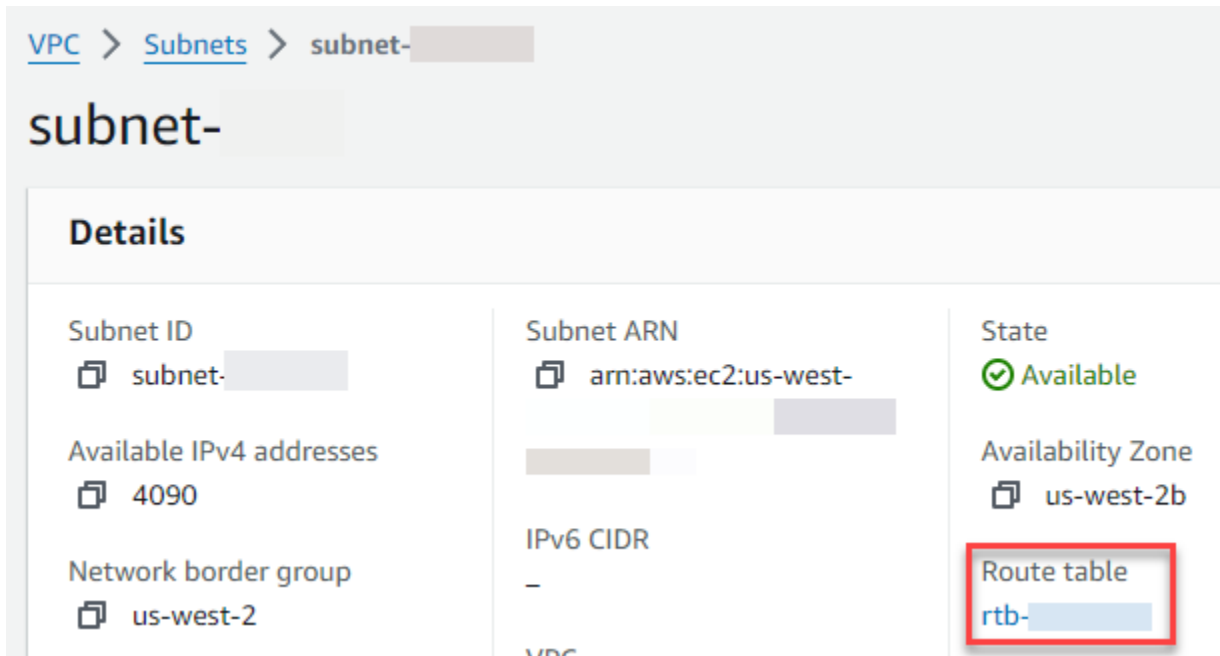
#### Note

Las puertas de enlace NAT están asociadas a una subred pública, pero la entrada de la tabla de enrutamiento se encuentra en la subred privada.

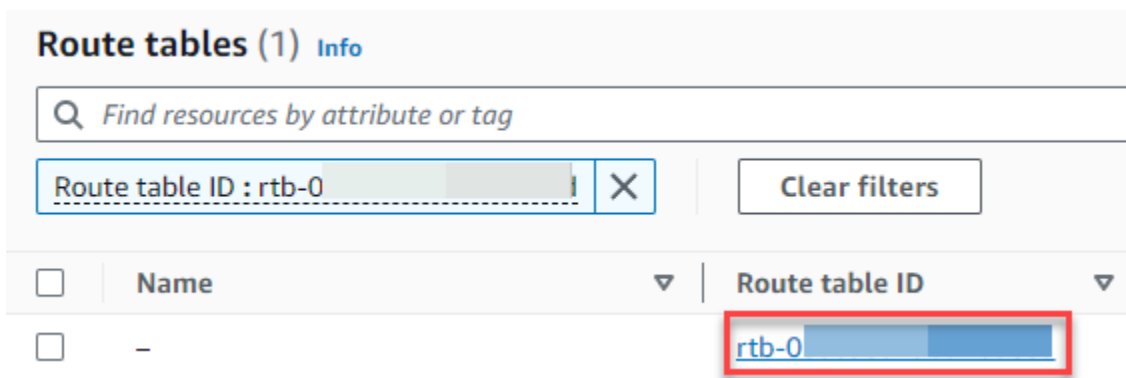
5. Para el ID de asignación de IP elástica, seleccione una dirección IP elástica o elija Asignar IP elástica.
6. Elija Crear una puerta de enlace de NAT.

## Adición de una ruta a la puerta de enlace NAT en la tabla de enrutamiento de la subred privada

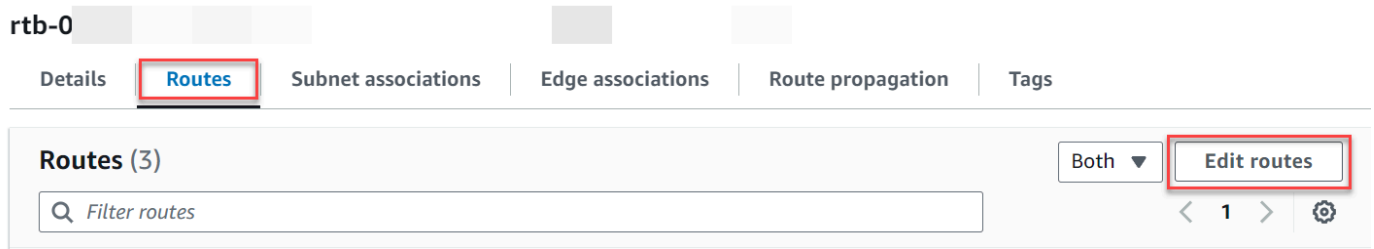
1. En el panel de navegación, elija Subnets (Subredes).
2. Seleccione una subred privada en su VPC. (La subred privada es una subred que no dispone de una ruta a una puerta de enlace de Internet en su tabla de enrutamiento).
3. Elija el enlace en Tabla de enrutamiento.



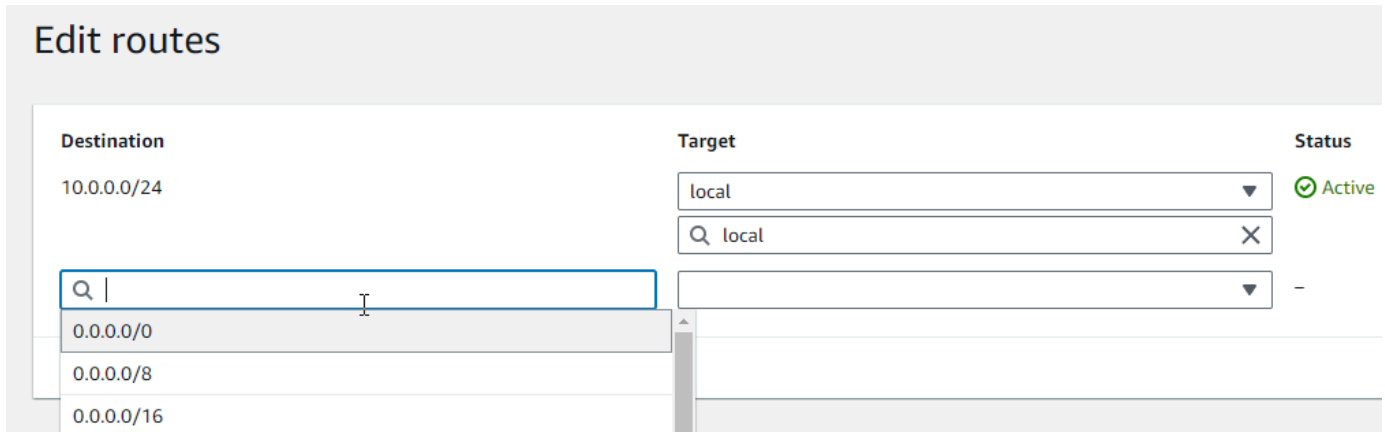
4. Elija el ID de la tabla de enrutamiento para abrir la página de detalles de la tabla de enrutamiento.



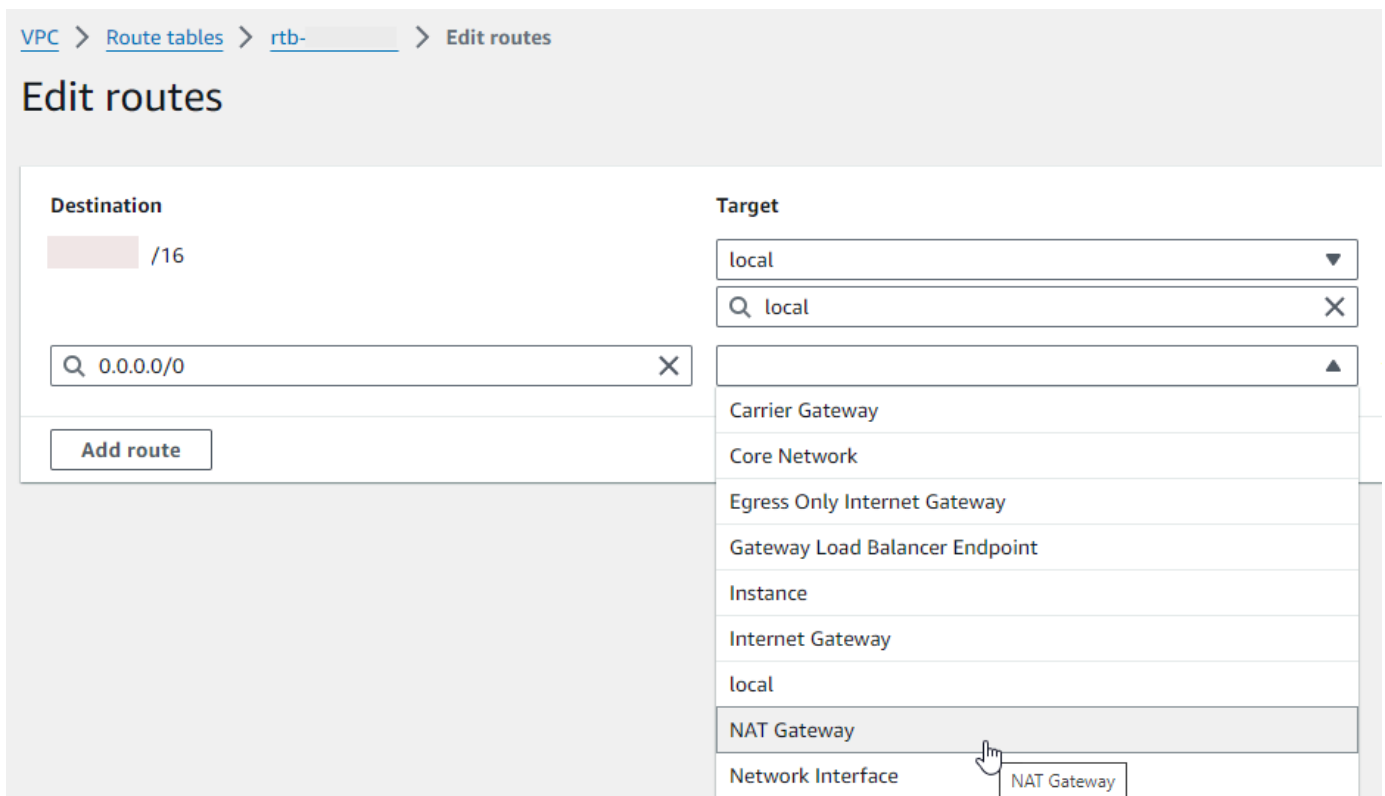
5. Desplácese hacia abajo y elija la pestaña Rutas y, a continuación, Editar rutas



6. Seleccione Agregar ruta y, a continuación, introduzca 0.0.0.0/0 en el cuadro Destino.



7. En Destino, seleccione Puerta de enlace NAT y, a continuación, elija la puerta de enlace NAT que creó anteriormente.



8. Elija Guardar cambios.



## Creación de una puerta de enlace de Internet de solo salida (solo IPv6)

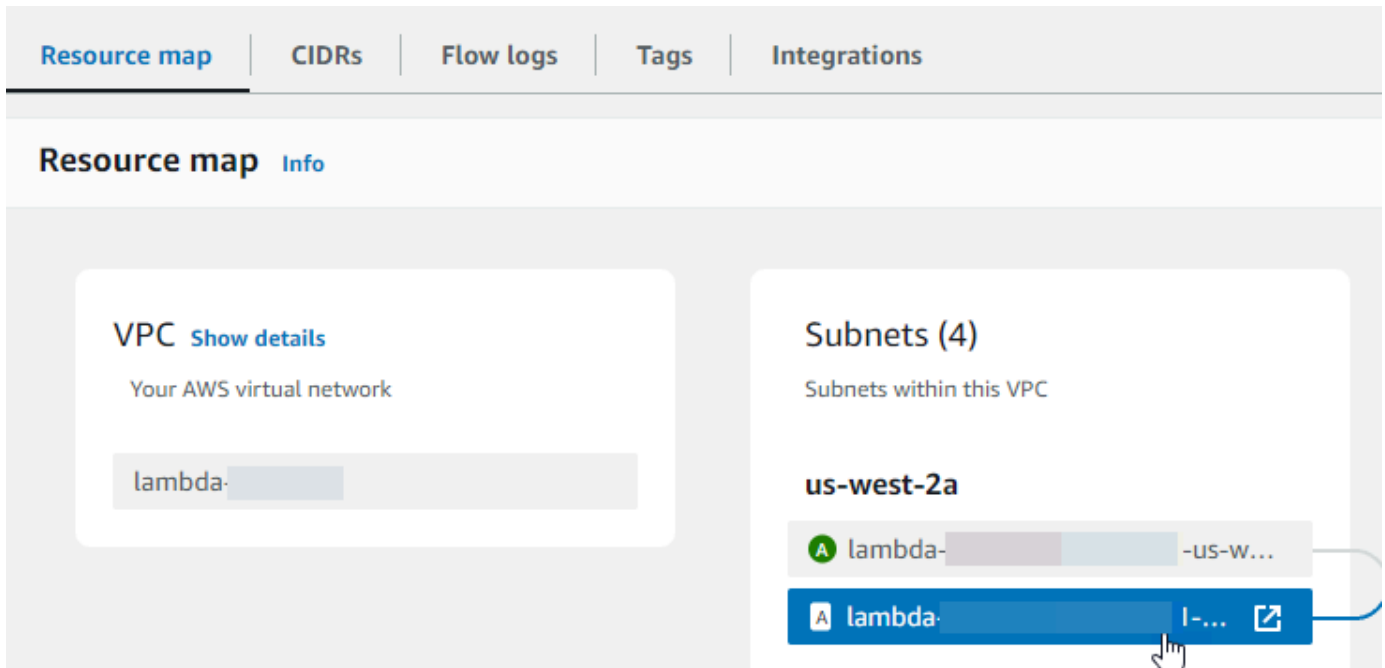
Siga estos pasos para crear una puerta de enlace de Internet de solo salida y agregar la tabla de enrutamiento de su subred privada.

Para crear una gateway de internet de solo salida

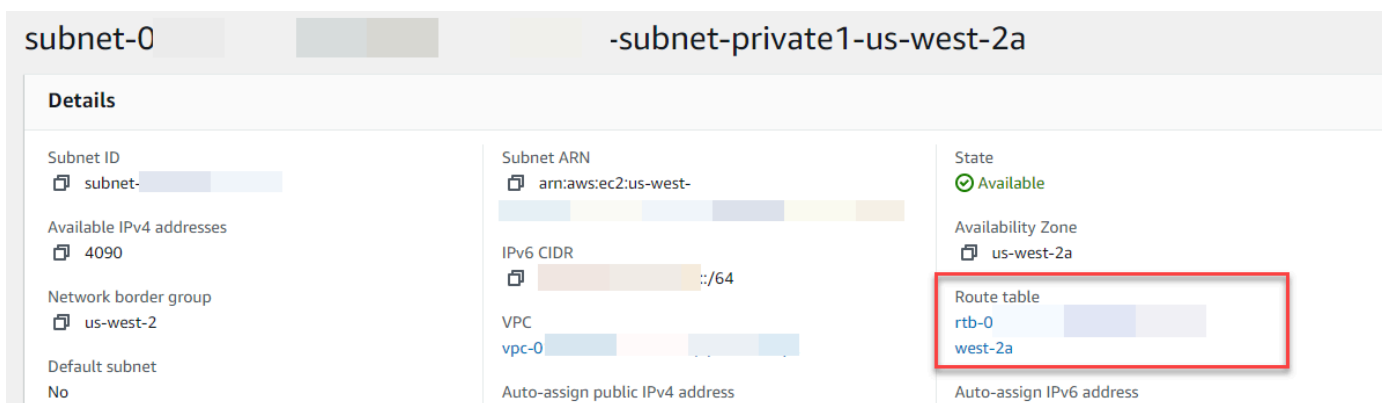
1. En el panel de navegación, elija Puertas de enlace de Internet de solo salida.
2. Elija Crear puerta de enlace de Internet de solo salida.
3. (Opcional) Escriba un nombre.
4. Seleccione la VPC en la que desea crear el puerto de enlace a Internet de solo salida.
5. Elija Crear puerta de enlace de Internet de solo salida.
6. Elija el enlace que aparece en el ID de VPC asociada.



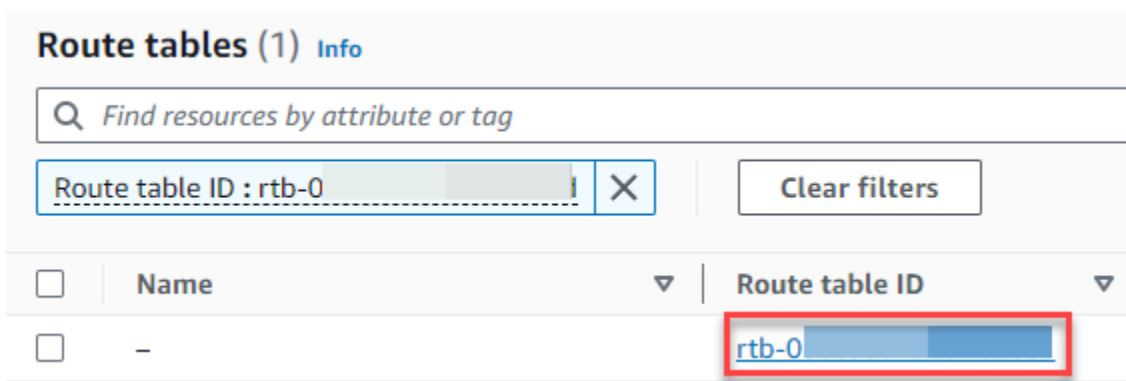
7. Seleccione el enlace en el ID de la VPC para abrir la página de detalles de VPC.
8. Desplácese hacia abajo hasta la sección Mapa de recursos y, a continuación, elija una subred privada. (La subred privada es una subred que no dispone de una ruta a una puerta de enlace de Internet en su tabla de enrutamiento). Los detalles de la subred se mostrarán en una nueva pestaña.



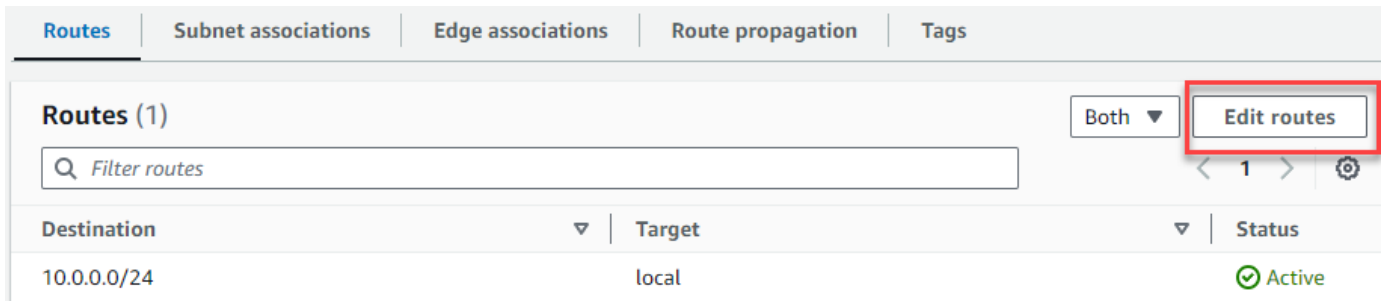
9. Elija el enlace en Tabla de enrutamiento.



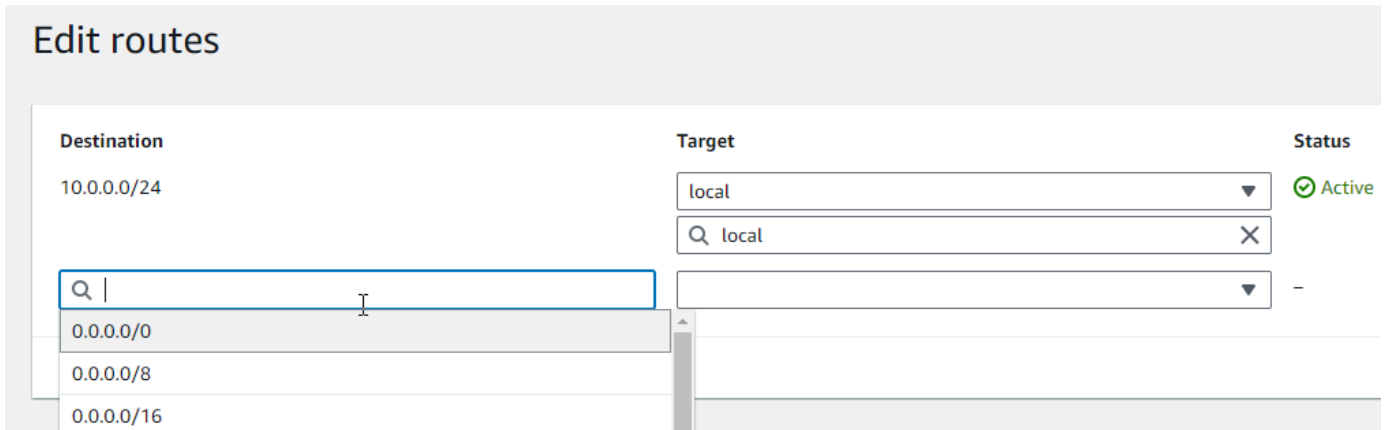
10. Elija el ID de la tabla de enrutamiento para abrir la página de detalles de la tabla de enrutamiento.



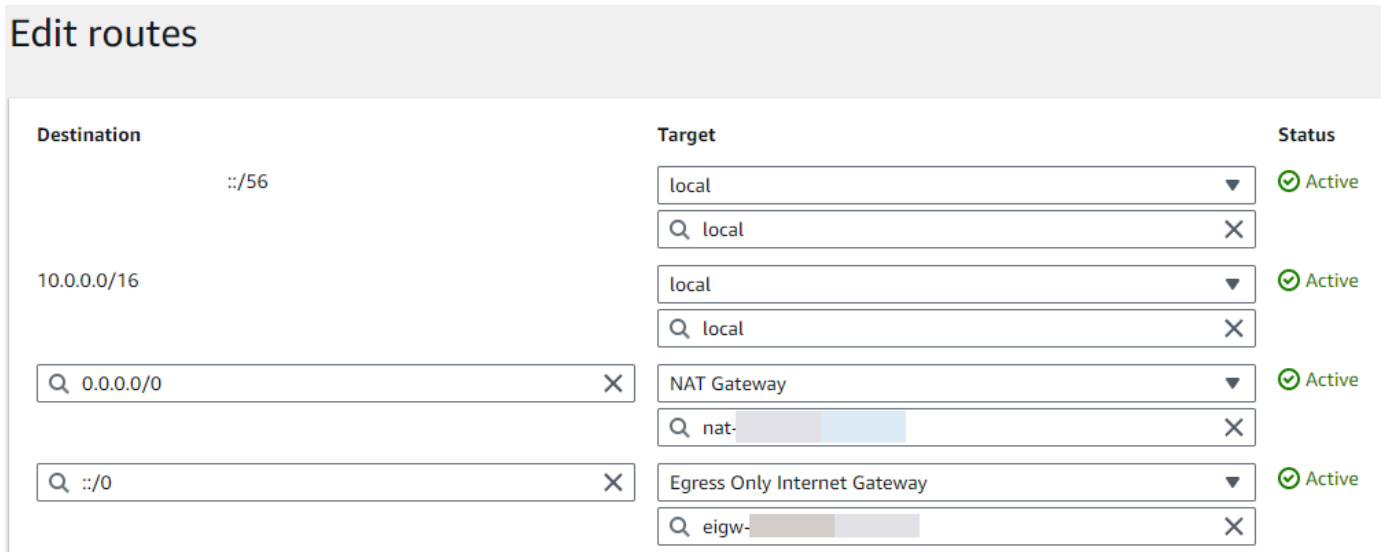
11. En Rutas, elija Editar rutas.



12. Seleccione Agregar ruta y, a continuación, introduzca `::/0` en el cuadro Destino.



13. En Destino, seleccione Puerta de enlace de Internet de solo salida y, a continuación, elija la puerta de enlace que creó anteriormente.



14. Elija Guardar cambios.

## Configuración de la función de Lambda

Para configurar una VPC al crear una función

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija Crear función.
3. En Basic information (Información básica), para Function name (Nombre de función), escriba un nombre para la función.
4. Amplíe Configuración avanzada.
5. Seleccione Habilitar VPC y, a continuación, elija una VPC.
6. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
7. En Subredes, seleccione todas las subredes privadas. Las subredes privadas pueden obtener acceso a Internet a través de una puerta de enlace NAT. La conexión de una función a una subred pública no le concede acceso a Internet.

### Note

Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.

8. En el caso de los Grupos de seguridad, seleccione un grupo de seguridad que permita el tráfico saliente.
9. Elija Crear función.

Lambda crea automáticamente un rol de ejecución con la política

[AWSLambdaVPCAccessExecutionRole](#) administrada por AWS. Los permisos de esta política solo son necesarios para crear interfaces de redes elásticas para la configuración de la VPC, no para invocar la función. Para aplicar permisos con privilegios mínimos, puede eliminar la política [AWSLambdaVPCAccessExecutionRole](#) de su rol de ejecución después de crear la función y la configuración de VPC. Para obtener más información, consulte [Permisos de IAM necesarios](#).

Para configurar una VPC para una función existente

Para agregar una configuración de VPC a una función existente, el rol de ejecución de la función debe tener [permiso para crear y administrar interfaces de redes elásticas](#). La política [AWSLambdaVPCAccessExecutionRole](#) administrada por AWS incluye los permisos

requeridos. Para aplicar permisos con privilegios mínimos, puede eliminar la política `AWSLambdaVPCAccessExecutionRole` de su rol de ejecución después de crear la configuración de VPC.

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Configuración y, a continuación, elija VPC.
4. En VPC, elija Edit (Editar).
5. Seleccione la VPC.
6. (Opcional) Para permitir el [tráfico IPv6 saliente](#), seleccione Permitir tráfico IPv6 para subredes de doble pila.
7. En Subredes, seleccione todas las subredes privadas. Las subredes privadas pueden obtener acceso a Internet a través de una puerta de enlace NAT. La conexión de una función a una subred pública no le concede acceso a Internet.

#### Note

Si seleccionó Permitir tráfico IPv6 para subredes de pila doble, todas las subredes seleccionadas deben tener un bloque de CIDR IPv4 y un bloque de CIDR IPv6.

8. En el caso de los Grupos de seguridad, seleccione un grupo de seguridad que permita el tráfico saliente.
9. Seleccione Guardar.

## Prueba de la función

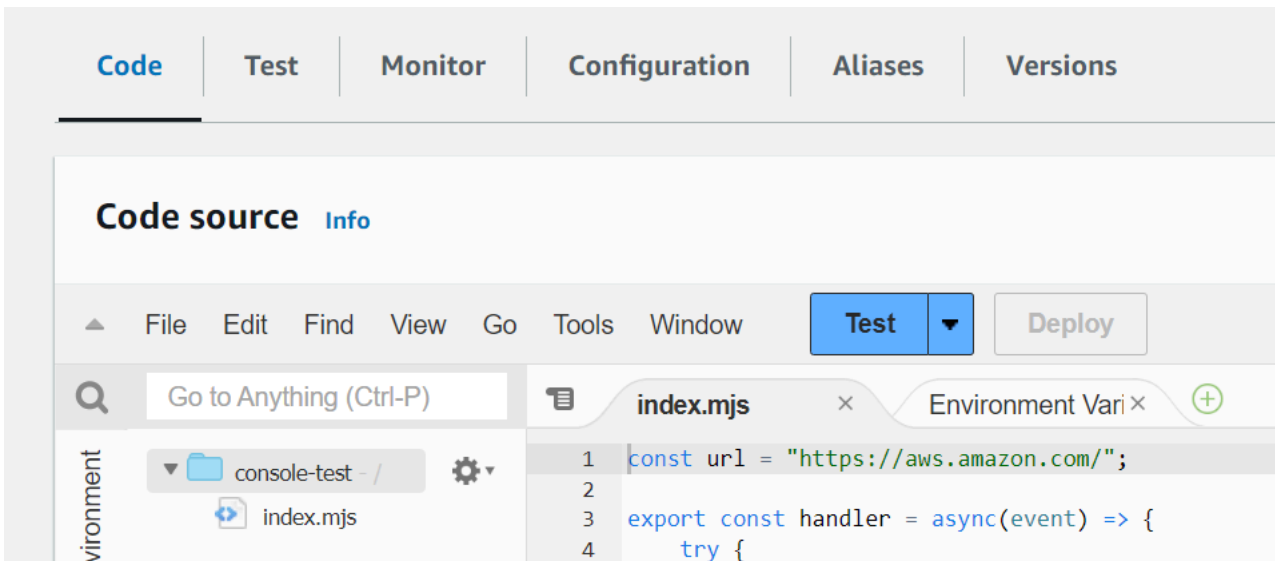
Use el siguiente código de ejemplo para confirmar que su función conectada a VPC puede llegar a la Internet pública. Si se ejecuta correctamente, el código devuelve un código de estado `200`. Si no funciona, la función agota el tiempo de espera.

### Node.js

En este ejemplo, se utiliza `fetch`, que está disponible en el tiempo de ejecución de `node.js18.x` y posteriores.

1. En el panel Código fuente de la consola de Lambda, pegue el siguiente código en el archivo `index.mjs`. La función realiza una solicitud HTTP GET a un punto de conexión público y

devuelve el código de respuesta HTTP para comprobar si la función tiene acceso a la Internet pública.

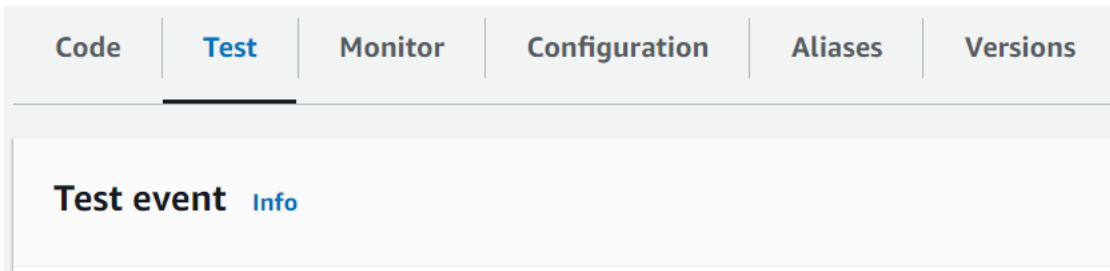


#### Example — Solicitud HTTP con async/await

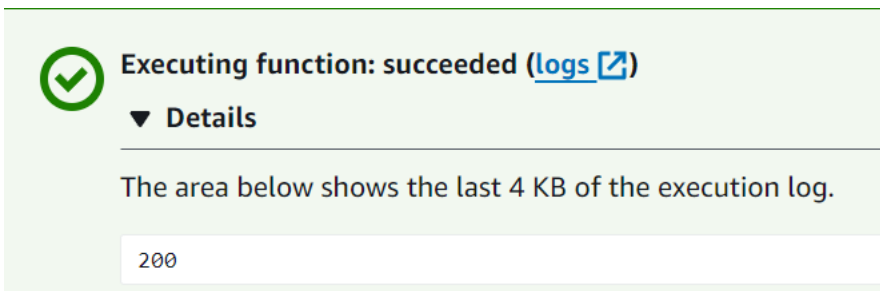
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
  try {
    // fetch is available with Node.js 18 and later runtimes
    const res = await fetch(url);
    console.info("status", res.status);
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

2. Elija Implementar.
3. Elija la pestaña Prueba.



4. Seleccione Probar.
5. La función devuelve un código de estado 200. Esto significa que la función tiene acceso saliente a Internet.

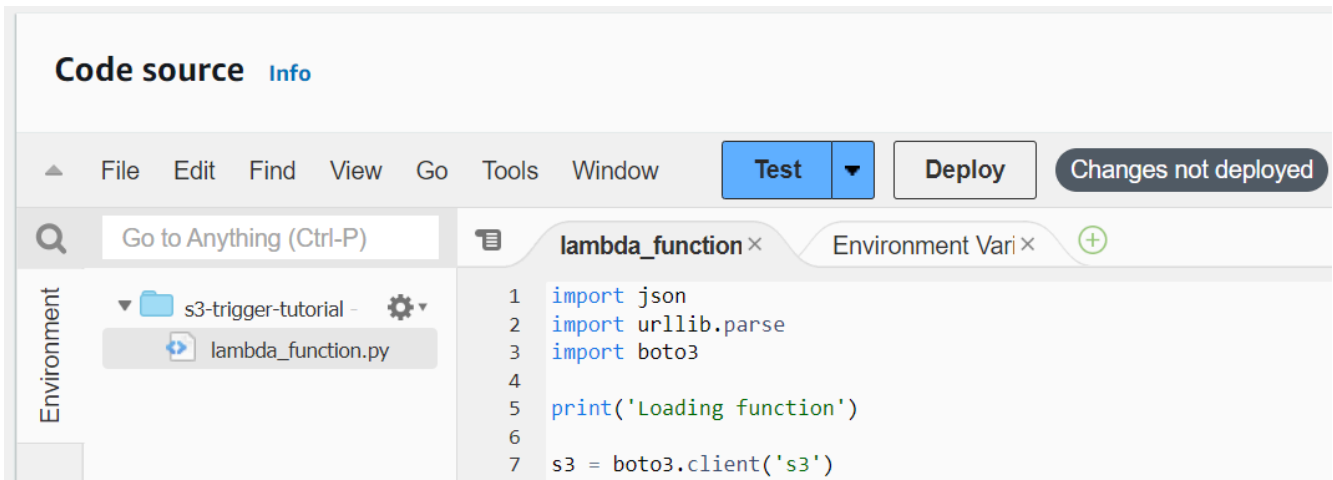


Si la función no puede acceder a la Internet pública, aparecerá un mensaje de error como este:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12j1c-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

## Python

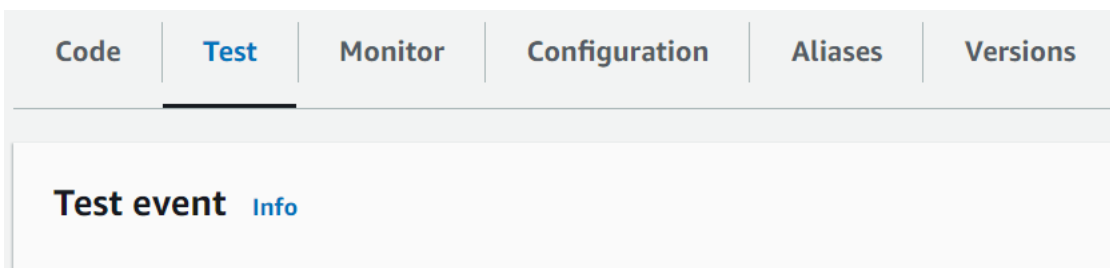
1. En el panel Código fuente de la consola de Lambda, pegue el siguiente código en el archivo `lambda_function.py`. La función realiza una solicitud HTTP GET a un punto de conexión público y devuelve el código de respuesta HTTP para comprobar si la función tiene acceso a la Internet pública.



```
import urllib.request

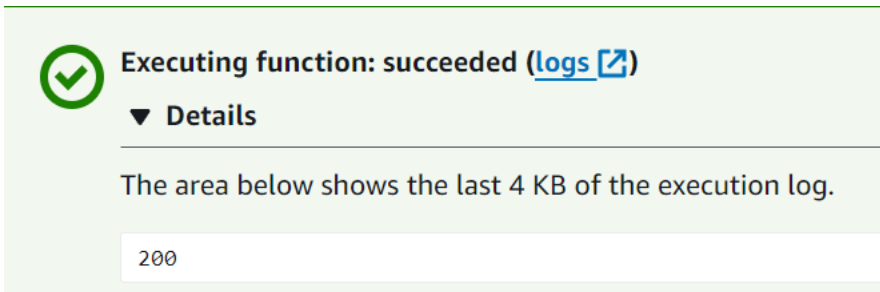
def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

2. Elija Implementar.
3. Elija la pestaña Prueba.



4. Seleccione Probar.
5. La función devuelve un código de estado 200. Esto significa que la función tiene acceso saliente a Internet.





The screenshot shows a green status bar with a checkmark icon and the text "Executing function: succeeded (logs [link])". Below this is a "Details" section with a downward arrow. A text box indicates "The area below shows the last 4 KB of the execution log." and a scrollable area displays the log content "200".

Si la función no puede acceder a la Internet pública, aparecerá un mensaje de error como este:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
Task timed out after 3.01 seconds"
}
```

# Conexión de puntos de conexión de VPC de la interfaz de entrada para Lambda

Si utiliza Amazon Virtual Private Cloud (Amazon VPC) para alojar sus recursos de AWS, puede establecer una conexión entre su VPC y Lambda. Puede utilizar esta conexión para invocar la función de Lambda sin pasar por la red pública de Internet.

Puede establecer una conexión privada entre la VPC y Lambda mediante la creación de un [punto de enlace de la VPC de interfaz](#). Los puntos de enlace de interfaz cuentan con la tecnología de [AWS PrivateLink](#), lo que les permite acceder de forma privada a las API de Lambda sin utilizar una puerta de enlace de Internet, un dispositivo NAT, una conexión de VPN o una conexión AWS Direct Connect. Las instancias de la VPC no necesitan direcciones IP públicas para comunicarse con las API de Lambda. El tráfico entre la VPC y Lambda no sale de la red de AWS.

Cada punto de enlace de la interfaz está representado por una o más [interfaces de redes elásticas](#) en las subredes. Una interfaz de red proporciona una dirección IP privada que sirve como punto de entrada del tráfico dirigido a Lambda.

## Secciones

- [Consideraciones para los puntos de enlace de la interfaz Lambda](#)
- [Creación de un punto de enlace de interfaz para Lambda](#)
- [Creación de una política de punto de enlace de interfaz para Lambda](#)

## Consideraciones para los puntos de enlace de la interfaz Lambda

Antes de configurar un punto de enlace de la interfaz para Lambda, revise el tema [Propiedades y limitaciones de los puntos de enlace de interfaz](#) en la Guía del usuario de Amazon VPC.

Puede llamar a cualquiera de las operaciones de API de Lambda desde su VPC. Por ejemplo, puede invocar la función de Lambda llamando a la API de Invoke desde su VPC. Para ver la lista completa de las API de Lambda, consulte [Actions \(Acciones\)](#) en la Referencia de las API de Lambda.

use1-az3 es una región de capacidad limitada para las funciones de la VPC de Lambda. No debe utilizar subredes en esta zona de disponibilidad con las funciones de Lambda, ya que esto puede reducir la redundancia zonal en caso de una interrupción.

## Directiva "keep-alive" para conexiones persistentes

Con el tiempo, Lambda depura las conexiones inactivas, por lo que es necesario utilizar una directiva keep-alive para conservar las conexiones persistentes. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#) en la Guía del desarrollador de AWS SDK for JavaScript.

## Consideraciones de facturación

No se generan costos adicionales por acceder a una función de Lambda a través de un punto de enlace. Para obtener más información sobre los precios de Lambda, consulte los [Precios de AWS Lambda](#).

Los precios estándar de AWS PrivateLink se aplica a los puntos de enlace de la interfaz para Lambda. Su cuenta de AWS se facturará cada hora de aprovisionamiento de un punto de enlace de interfaz en cada zona de disponibilidad y los datos procesados a través del punto de enlace de la interfaz. Para obtener más información sobre los precios de los puntos de enlace de tipo interfaz, consulte [Precios de AWS PrivateLink](#).

## Consideraciones sobre la interconexión de VPC

Puede conectar una VPC a otra con puntos de enlace de interfaz mediante [Interconexión con VPC](#). El emparejamiento de VPC es una conexión de red entre dos VPC. Puede establecer una interconexión de VPC entre dos VPC propias o con una VPC de otra cuenta de AWS. Las VPC también pueden estar en dos regiones de AWS diferentes.

El tráfico entre las VPC emparejadas permanece en la red de AWS y no pasa por la red pública de Internet. Una vez que las VPC están interconectadas, algunos recursos como las instancias de Amazon Elastic Compute Cloud (Amazon EC2), las instancias de Amazon Relational Database Service (Amazon RDS) o las funciones de Lambda habilitadas para VPC pueden acceder a la API de Lambda a través de puntos de enlace de interfaz creados en una de las VPC.

## Creación de un punto de enlace de interfaz para Lambda

Puede crear un punto de enlace de interfaz para Lambda mediante la consola de Amazon VPC o la AWS Command Line Interface (AWS CLI). Para más información, consulte [Creación de un punto de conexión de interfaz](#) en la Guía del usuario de Amazon VPC.

Para crear un punto de enlace de interfaz para Lambda (consola)

1. Abra la página [Endpoints \(Puntos de enlace\)](#) de la consola de Amazon VPC.
2. Seleccione Crear punto de conexión.
3. En Service category (Categoría del servicio), asegúrese de que se seleccione servicios de AWS.
4. En Service Name (Nombre del servicio), elija `com.amazonaws.región.lambda`. Compruebe que el valor de Type (Tipo) es Interface (Interfaz).
5. Elija una VPC y las subredes.
6. Para habilitar un DNS privado para el punto de enlace de interfaz, seleccione Enable DNS Name (Habilitar nombre de DNS), en la casilla de verificación. Le recomendamos que habilite nombres DNS privados para su punto de conexión de VPC para los Servicios de AWS. Esto garantiza que las solicitudes que utilizan los puntos de conexión de servicio público, como las solicitudes realizadas a través de un SDK de AWS, se resuelvan en su punto de conexión de VPC.
7. En Security group (Grupo de seguridad), elija uno o varios grupos de seguridad.
8. Seleccione Crear punto de conexión.

Para poder utilizar la opción de DNS privado, debe definir `enableDnsHostnames` y `enableDnsSupportattributes` en su VPC. Para obtener más información, consulte [Viewing and updating DNS support for your VPC \(Visualización y actualización de la compatibilidad de DNS para su VPC\)](#) en la Guía del usuario de Amazon VPC. Si habilita DNS privado para el punto de enlace de interfaz, puede realizar solicitudes a la API para Lambda usando su nombre de DNS predeterminado para la región, por ejemplo `lambda.us-east-1.amazonaws.com`. Para ver otros puntos de enlace de servicio, consulte [Service endpoints and quotas](#) en la Referencia general de AWS.

Para más información, consulte [Acceso a un servicio a través de un punto de conexión de interfaz](#) en la Guía del usuario de Amazon VPC.

Para obtener información acerca de cómo se crea y configura un punto de enlace mediante AWS CloudFormation, consulte el recurso [AWS::EC2::VPCEndpoint](#) en la Guía del usuario de AWS CloudFormation.

Si desea crear un punto de enlace de la interfaz para Lambda (AWS CLI)

Utilice el comando `create-vpc-endpoint` y especifique el ID de la VPC, el tipo de punto de enlace de la VPC (interfaz), el nombre del servicio, las subredes que usarán el punto de enlace y los grupos de seguridad que se asociarán a las interfaces de red del punto de enlace. Por ejemplo:

```
aws ec2 create-vpc-endpoint --vpc-id vpc-ec43eb89 --vpc-endpoint-type Interface --
service-name \
    com.amazonaws.us-east-1.lambda --subnet-id subnet-abababab --security-group-id
sg-1a2b3c4d
```

## Creación de una política de punto de enlace de interfaz para Lambda

Para controlar quién puede usar en punto de enlace de la interfaz y a qué funciones de Lambda tiene acceso el usuario, puede asociar una política al punto de enlace. La política especifica la siguiente información:

- La entidad principal que puede realizar acciones.
- Acciones que la entidad principal puede realizar.
- Recursos en los que la entidad principal puede realizar acciones.

Para más información, consulte [Control del acceso a los servicios con puntos de enlace de la VPC](#) en la Guía del usuario de Amazon VPC.

Ejemplo: Política de puntos de enlace de tipo interfaz para acciones de Lambda

A continuación, se muestra un ejemplo de una política de puntos de enlace de Lambda. Cuando la política se asocia a un punto de enlace, permite que el usuario `MyUser` invoque la función `my-function`.

### Note

Debe incluir el ARN completo e incompleto de la función en el recurso.

```
{
  "Statement": [
    {
      "Principal":
        {
          "AWS": "arn:aws:iam::111122223333:user/MyUser"
        },
      "Effect": "Allow",
```

```
    "Action": [
      "lambda:InvokeFunction"
    ],
    "Resource": [
      "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
    ]
  }
]
}
```

# Configuración del acceso al sistema de archivos para las funciones de Lambda

Puede configurar una función para montar un sistema de archivos Amazon Elastic File System (Amazon EFS) a un directorio local. Con Amazon EFS, el código de función puede acceder y modificar los recursos compartidos de forma segura y en alta concurrencia.

## Secciones

- [Permisos de usuario y rol de ejecución](#)
- [Configuración de un sistema de archivos y un punto de acceso](#)
- [Conexión a un sistema de archivos \(consola\)](#)

## Permisos de usuario y rol de ejecución

Si el sistema de archivos no tiene una política de AWS Identity and Access Management (IAM) configurada por el usuario, EFS utiliza una política predeterminada que otorga acceso completo a cualquier cliente que pueda conectarse al sistema de archivos mediante un destino de montaje del sistema de archivos. Si el sistema de archivos tiene una política de IAM configurada por el usuario, el rol de ejecución de la función debe tener los permisos de `elasticfilesystem` correctos.

### Permisos de rol de ejecución

- `elasticfilesystem:ClientMount`
- `elasticfilesystem:ClientWrite` (no se necesita para conexiones de solo lectura)

Estos permisos se incluyen en la política administrada `AmazonElasticFileSystemClientReadWriteAccess`. Además, el rol de ejecución debe tener los [permisos necesarios para conectarse a la VPC del sistema de archivos](#).

Cuando configura un sistema de archivos, Lambda utiliza sus permisos para verificar los destinos de montaje. Para configurar una función para conectarse a un sistema de archivos, el usuario necesita los siguientes permisos:

### Permisos de usuario

- `elasticfilesystem:DescribeMountTargets`

## Configuración de un sistema de archivos y un punto de acceso

Cree un sistema de archivos en Amazon EFS con un destino de montaje en cada zona de disponibilidad a la que se conecte su función. Para obtener rendimiento y resistencia, utilice al menos dos zonas de disponibilidad. Por ejemplo, en una configuración simple, podría tener una VPC con dos subredes privadas en zonas de disponibilidad separadas. La función se conecta a ambas subredes y un destino de montaje está disponible en cada una. Asegúrese de que los grupos de seguridad utilizados por la función y los destinos de montaje permiten el tráfico NFS (puerto 2049).

### Note

Cuando crea un sistema de archivos, elige un modo de rendimiento que no se puede cambiar más adelante. El modo Propósito general tiene menor latencia, y el modo E/S Max admite un rendimiento e IOPS máximos más altos. Para obtener ayuda sobre cómo elegir, consulte [Rendimiento de Amazon EFS](#) en la Guía del usuario de Amazon Elastic File System.

Un punto de acceso conecta cada instancia de la función con el destino de montaje correcto para la zona de disponibilidad a la que se conecta. Para obtener el mejor rendimiento, cree un punto de acceso con una ruta que no sea raíz y limite el número de archivos que crea en cada directorio. En el siguiente ejemplo se crea un directorio denominado `my-function` en el sistema de archivos y se establece el identificador de propietario en 1001 con permisos de directorio estándar (755).

### Example configuración de punto de acceso

- Nombre: `files`
- ID de usuario: `1001`
- ID del grupo : `1001`
- Ruta – `/my-function`
- Permisos: `755`
- ID de usuario del propietario – `1001`
- ID de usuario de grupo – `1001`

Cuando una función utiliza el punto de acceso, se le da el ID de usuario 1001 y tiene acceso completo al directorio.



Para obtener más información, consulte los siguientes temas en la Guía del usuario de Amazon Elastic File System:

- [Creación de recursos para Amazon EFS](#)
- [Trabajar con usuarios, grupos y permisos](#)

## Conexión a un sistema de archivos (consola)

Una función se conecta a un sistema de archivos a través de la red local en una VPC. Las subredes a las que se conecta la función pueden ser las mismas subredes que contienen puntos de montaje para el sistema de archivos, o subredes en la misma zona de disponibilidad que pueden enrutar el tráfico NFS (puerto 2049) al sistema de archivos.

### Note

Si la función aún no está conectada a una VPC, consulte [Otorgamiento a las funciones de Lambda de acceso a los recursos de una Amazon VPC](#).

Para configurar el acceso al sistema de archivos

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuración y, a continuación, elija Sistemas de archivos.
4. En Sistema de archivos, elija Agregar sistema de archivos.
5. Configure las siguientes propiedades:
  - Sistema de archivos EFS: el punto de acceso para un sistema de archivos de la misma VPC.
  - Ruta de montaje local: la ubicación en la que se monta el sistema de archivos en la función de Lambda, empezando por `/mnt/`.

### Precios

Amazon EFS tiene cargos por almacenamiento y rendimiento, con tarifas que varían según la clase de almacenamiento. Para obtener más información, consulte [Precios de Amazon EFS](#).

Cargos de Lambda por transferencia de datos entre VPC. Esto solo se aplica si la VPC de su función está pegada a otra VPC con un sistema de archivos. Las tasas son las mismas que para la transferencia de datos de Amazon EC2 entre VPC de la misma región. Para obtener más información, consulte [Precios de Lambda](#).

# Crear un alias para una función de Lambda

Puede crear alias para una función de Lambda. Un alias de Lambda es un puntero a una versión de la función que se puede actualizar. Los usuarios de la función pueden acceder a la versión de la función utilizando el nombre de recurso de Amazon (ARN) del alias. Cuando se implementa una versión nueva, se puede actualizar el alias para usar la nueva versión o dividir el tráfico entre dos versiones.

## Console

Para crear un alias mediante la consola

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Alias y, a continuación, elija Crear alias.
4. En la página Crear alias, haga lo siguiente:
  - a. En Nombre, escriba el nombre del alias.
  - b. (Opcional) En Descripción, escriba una descripción del alias.
  - c. En Versión, elija la versión de la función a la que desee que apunte el alias.
  - d. (Opcional) Para configurar el direccionamiento del alias, expanda Weighted alias (Alias ponderado). Para obtener más información, consulte [Crear una configuración de direccionamiento para un alias de Lambda](#).
  - e. Elija Save (Guardar).

## AWS CLI

Para crear un alias a través de AWS Command Line Interface (AWS CLI), utilice el comando [create-alias](#).

```
aws lambda create-alias \  
  --function-name my-function \  
  --name alias-name \  
  --function-version version-number \  
  --description " "
```

Si desea cambiar un alias para que apunte a una nueva versión de la función, utilice el comando [update-alias](#).

```
aws lambda update-alias \  
  --function-name my-function \  
  --name alias-name \  
  --function-version version-number
```

Para eliminar un alias, utilice el comando [delete-alias](#).

```
aws lambda delete-alias \  
  --function-name my-function \  
  --name alias-name
```

Los comandos de la AWS CLI mencionados en los pasos anteriores se corresponden con las siguientes operaciones de la API de Lambda:

- [CreateAlias](#)
- [UpdateAlias](#)
- [DeleteAlias](#)

## Uso de alias de Lambda en las fuentes de eventos y las políticas de permisos

Cada alias tiene un ARN único. Un alias solo puede apuntar a una versión de una función, no a otro alias. Puede actualizar un alias para que apunte a una nueva versión de la función.

Las fuentes de eventos como Amazon Simple Storage Service (Amazon S3) invocan su función Lambda. Estos orígenes de eventos mantienen un mapeo que identifica la función que invocarán cuando se produzcan eventos. Si especifica un alias de función de Lambda en la configuración de mapeo, no es necesario actualizar el mapeo cuando cambie la versión de la función. Para obtener más información, consulte [Cómo procesa Lambda registros de orígenes de eventos basados en secuencias y colas](#).

En una política de recursos, puede conceder permisos para que las fuentes de eventos utilicen la función de Lambda. Si especifica un ARN de alias en la política, no es necesario que actualice la política cuando cambie la versión de la función.

### Políticas de recursos

Puede utilizar una [política basada en recursos](#) para proporcionar a la función acceso a un servicio, recurso o cuenta. El ámbito de ese permiso dependerá de si se aplica a un alias, a una versión o a toda la función. Por ejemplo, si utiliza un nombre de alias (como `helloworld:PROD`), el permiso le permitirá invocar la función `helloworld` utilizando el ARN del alias (`helloworld:PROD`).

Si intenta invocar la función sin un alias o una versión específica, se producirá un error con los permisos. Este error de los permisos tendrá lugar aunque intente invocar directamente la versión de la función asociada al alias.

Por ejemplo, el siguiente comando de AWS CLI otorga permisos a Amazon S3 para invocar el alias `PROD` de la función `helloworld` cuando Amazon S3 actúa en nombre de `amzn-s3-demo-bucket`.

```
aws lambda add-permission \  
  --function-name helloworld \  
  --qualifier PROD \  
  --statement-id 1 \  
  --principal s3.amazonaws.com \  
  --action lambda:InvokeFunction \  
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \  
  --source-account 123456789012
```

Para obtener más información sobre el uso de nombres de recursos en las políticas, consulte [Afinar las secciones de recursos y condiciones de las políticas](#).

## Crear una configuración de direccionamiento para un alias de Lambda

Utilice la configuración de direccionamiento en un alias para enviar una parte del tráfico a una segunda versión de la función. Por ejemplo, puede reducir el riesgo de implementar una nueva versión configurando el alias para enviar la mayor parte del tráfico a la versión existente y solo un pequeño porcentaje del tráfico a la nueva versión.

Lambda utiliza un modelo probabilístico simple para distribuir el tráfico entre las dos versiones de funciones. En niveles de tráfico bajos, es posible que vea una gran variación entre el porcentaje configurado y el porcentaje real de tráfico en cada versión. Si su función usa la concurrencia aprovisionada, puede evitar [Invocaciones de superación](#) configurando un mayor número de instancias de simultaneidad aprovisionadas durante el tiempo en que el enrutamiento de alias está activo.

Puede apuntar un alias a un máximo de dos versiones de una función de Lambda. Las versiones deben cumplir los siguientes criterios:

- Ambas versiones deben tener el mismo [rol de ejecución](#).
- Ambas versiones deben tener la misma configuración de [cola de mensajes fallidos](#) o ninguna configuración de cola de mensajes fallidos.
- Ambas versiones deben publicarse. El alias no puede apuntar a \$LATEST.

### Console

Para configurar el direccionamiento en un alias con la consola

#### Note

Verifique que la función tenga al menos dos versiones publicadas. Si necesita crear otras versiones, siga las instrucciones que se indican en [Administrar las versiones de la función de Lambda](#).

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.

3. Elija Alias y, a continuación, elija Crear alias.
4. En la página Crear alias, haga lo siguiente:
  - a. En Nombre, escriba el nombre del alias.
  - b. (Opcional) En Descripción, escriba una descripción del alias.
  - c. En Versión, elija la primera versión de la función a la que desea que apunte el alias.
  - d. Expanda Weighted alias (Alias ponderados).
  - e. En Versión adicional, elija la segunda versión de la función a la que desea que apunte el alias.
  - f. En Weight (%) [Ponderación (%)], especifique la ponderación de la función. La ponderación es el porcentaje de tráfico que se asigna a dicha versión cuando se invoca el alias. La primera versión recibe la ponderación residual. Por ejemplo, si especifica un 10 por ciento en Additional version (Versión adicional), a la primera versión se asigna automáticamente el 90 %.
  - g. Seleccione Guardar.

## AWS CLI

Utilice los comandos [create-alias](#) y [update-alias](#) de AWS CLI para configurar las ponderaciones de tráfico entre dos versiones de funciones. Cuando se crea o actualiza el alias, se especifica la ponderación del tráfico en el parámetro `routing-config`.

En el ejemplo siguiente se crea un alias de una función de Lambda llamado `routing-alias` que apunta a la versión 1 de la función. La versión 2 de la función recibe el 3 por ciento del tráfico. El 97 por ciento restante del tráfico se dirige a la versión 1.

```
aws lambda create-alias \  
  --name routing-alias \  
  --function-name my-function \  
  --function-version 1 \  
  --routing-config AdditionalVersionWeights={"2":0.03}
```

Utilice el comando `update-alias` para aumentar el porcentaje de tráfico entrante a la versión 2. En el ejemplo siguiente, aumenta el tráfico al 5 por ciento.

```
aws lambda update-alias \  
  --name routing-alias \  
  --routing-config AdditionalVersionWeights={"2":0.05}
```

```
--function-name my-function \  
--routing-config AdditionalVersionWeights={"2"]=0.05}
```

Si desea dirigir todo el tráfico a la versión 2, utilice el comando `update-alias` para cambiar la propiedad `function-version` y dirigir el alias a la versión 2. El comando también restablece la configuración de direccionamiento.

```
aws lambda update-alias \  
  --name routing-alias \  
  --function-name my-function \  
  --function-version 2 \  
  --routing-config AdditionalVersionWeights={}
```

Los comandos de la AWS CLI mencionados en los pasos anteriores se corresponden con las siguientes operaciones de la API de Lambda:

- [CreateAlias](#)
- [UpdateAlias](#)

## Cómo saber qué versión se invocó

Al configurar las ponderaciones de tráfico entre dos versiones de función, hay dos formas de determinar la versión de la función de Lambda que se ha invocado:

- CloudWatch Logs: Lambda emite automáticamente una entrada de log START que contiene el ID de la versión invocada a Amazon CloudWatch Logs para cada invocación de función. A continuación, se muestra un ejemplo:

```
19:44:37 START RequestId: request id Version: $version
```

En las invocaciones de alias, Lambda utiliza la dimensión `Executed Version` para filtrar los datos de las métricas por la versión ejecutada. Para obtener más información, consulte [Ver las métricas de funciones de Lambda](#).

- Carga de respuesta (invocaciones sincrónicas): las respuestas a invocaciones de funciones sincrónicas incluyen un encabezado `x-amz-executed-version` para indicar qué versión de función se ha invocado.



# Administrar las versiones de la función de Lambda

Puede usar versiones para administrar la implementación de funciones. Por ejemplo, puede publicar una nueva versión de una función para pruebas beta sin afectar a los usuarios de la versión de producción estable. Lambda crea una nueva versión de la función de cada vez que se publica la función. La nueva versión es una copia de la versión no publicada de la función. La versión no publicada se denomina \$LATEST.

## Note

Para crear una versión nueva de la función, primero debe realizar cambios en la versión no publicada (\$LATEST). Estos cambios pueden incluir la actualización del código o la modificación de la configuración. Si \$LATEST es idéntica a una versión publicada anteriormente, no podrá crear una nueva versión hasta que implemente los cambios en \$LATEST.

Tras publicar la versión de una función, su código, tiempo de ejecución, arquitectura, memoria, capas y la mayoría de las demás configuraciones son inmutables. Esto significa que no puede cambiar estas configuraciones sin publicar una nueva versión de \$LATEST. Puede configurar los siguientes elementos para una versión de función publicada:

- [Disparadores](#)
- [Destinos](#)
- [Simultaneidad aprovisionada](#)
- [Invocación asincrónica](#)
- [Conexiones y proxies de bases de datos](#)

## Note

Cuando se utilizan los [controles de administración del tiempo de ejecución](#) con el modo Automático, la versión del tiempo de ejecución utilizada por la versión de la función se actualiza automáticamente. Cuando se utilizan los modos Function update (Actualización de funciones) o Manual, no se actualiza la versión del tiempo de ejecución. Para obtener más información, consulte [the section called “Actualizaciones de las versiones del tiempo de ejecución”](#).

## Secciones

- [Creación de versiones de funciones](#)
- [Uso de versiones](#)
- [Concesión de permisos](#)

## Creación de versiones de funciones

Puede cambiar el código y la configuración de la función solo en la versión no publicada de una función. Cuando se publica una versión, Lambda bloquea el código y la mayoría de las opciones de configuración para garantizar una experiencia uniforme a los usuarios de dicha versión.

Puede crear una versión de función usando la consola de Lambda.

Para crear una nueva versión de función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función y, a continuación, elija Versiones.
3. En la página de configuración de versiones, elija Publicar nueva versión.
4. (Opcional) Especifique una descripción de la versión.
5. Elija Publicar.

También puede publicar una versión de una función con la operación de la API [PublishVersion](#).

El siguiente comando AWS CLI publica una nueva versión de una función. La respuesta devuelve la información de configuración sobre la nueva versión, incluido el número de la versión y el ARN de la función con el sufijo de la versión.

```
aws lambda publish-version --function-name my-function
```

Debería ver los siguientes datos de salida:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
  "Version": "1",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
```

```
"Handler": "function.handler",
"Runtime": "nodejs20.x",
...
}
```

### Note

Lambda asigna números de secuencia que aumentan de manera monótona para el control de versiones. Lambda nunca vuelve a utilizar los números de versión, ni siquiera después de eliminar y volver a crear una función.

## Uso de versiones

Puede hacer referencia a la función de Lambda utilizando un ARN completo o incompleto.

- ARN completo: ARN de la función con el sufijo de la versión. El siguiente ejemplo hace referencia a la versión 42 de la función `helloworld`.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:42
```

- ARN incompleto: ARN de la función sin el sufijo de la versión.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld
```

Puede utilizar un ARN completo o incompleto en todas las operaciones de API que corresponda. Sin embargo, no puede utilizar un ARN incompleto para crear un alias.

Si decide no publicar versiones de las funciones, puede utilizar tanto el ARN completo como incompleto en la [asignación de orígenes de eventos](#). Cuando se invoca una función usando un ARN incompleto, Lambda invoca \$LATEST implícitamente.

Lambda solo publica una nueva versión de la función si el código nunca se ha publicado o si este ha cambiado desde la última versión publicada. Si no hay ningún cambio, la versión de la función sigue siendo la que se publicó más recientemente.

El ARN completo de cada versión de una función de Lambda es único. Después de publicar una versión, no puede cambiar el ARN o el código de función.

## Concesión de permisos

Puede utilizar una [política basada en recursos](#) o una [política basada en identidades](#) para conceder acceso a la función. El ámbito del permiso dependerá de si la política se aplica a una función o a una versión de la función. Para obtener más información sobre los nombres de recursos de funciones en las políticas, consulte [Afinar las secciones de recursos y condiciones de las políticas](#).

Puede simplificar la administración de los orígenes de eventos y las políticas de AWS Identity and Access Management (IAM) utilizando un alias de función. Para obtener más información, consulte [Crear un alias para una función de Lambda](#).

# Uso de etiquetas en funciones de Lambda

Puede etiquetar funciones de AWS Lambda para activar el [control de acceso basado en atributos \(ABAC\)](#) y organizarlas por propietario, proyecto o departamento. Las etiquetas son pares clave-valor de forma libre que se admiten en todos los servicios de AWS para utilizarlas en ABAC, filtrar recursos y [agregar detalles a los informes de facturación](#).

Las etiquetas se aplican en el nivel de las funciones, no de las versiones ni de los alias. Las etiquetas no forman parte de la configuración específica de la versión que Lambda crea una instantánea al publicar una versión.

## Secciones

- [Permisos necesarios para trabajar con etiquetas](#)
- [Uso de etiquetas con la consola de Lambda](#)
- [Uso de etiquetas con la AWS CLI](#)
- [Requisitos para etiquetas](#)

## Permisos necesarios para trabajar con etiquetas

Conceda los permisos adecuados a la identidad de AWS Identity and Access Management (IAM) (usuario, grupo o rol) para la persona que trabaja con la función:

- `lambda: ListTags` — Cuando una función tiene etiquetas, concede este permiso a cualquier persona que necesite llamarla `GetFunction` o `ListTags` utilizarla.
- `lambda: TagResource` — Concede este permiso a cualquier persona que necesite llamar `CreateFunction` o `TagResource`

Para obtener más información, consulte [Políticas de IAM basadas en identidad para Lambda](#).

## Uso de etiquetas con la consola de Lambda

Puede utilizar la consola de Lambda para crear funciones que tengan etiquetas, agregar etiquetas a funciones existentes y filtrar funciones por las etiquetas que se agregan.

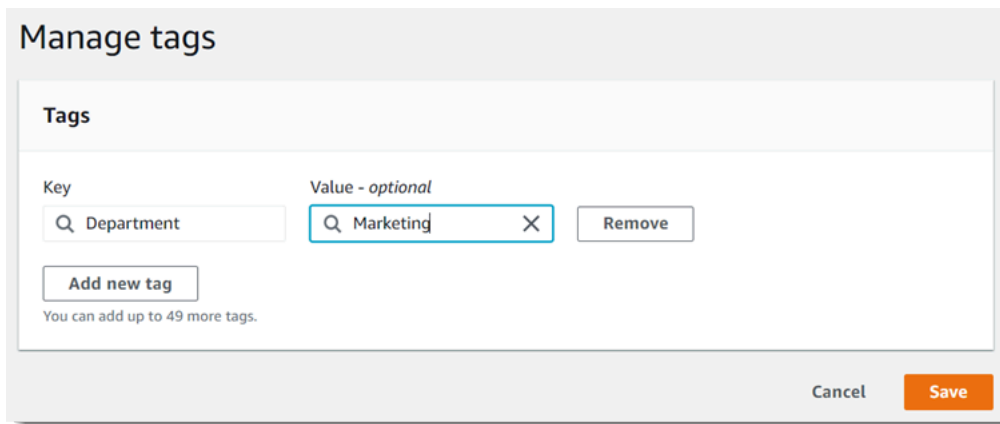
Para agregar etiquetas al momento de crear una función

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.

2. Elija Crear función.
3. Elija Author from scratch (Crear desde cero) o Container image (Imagen de contenedor).
4. Bajo Basic information (Información básica), haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función. Los nombres de las funciones están limitados a 64 caracteres de longitud.
  - b. Para Runtime (Tiempo de ejecución), elija la versión del idioma que desea utilizar para su función.
  - c. (Opcional) En Architecture (Arquitectura), elija la [arquitectura del conjunto de instrucciones](#) que utilizará para su función. La arquitectura predeterminada es x86\_64. Cuando cree el paquete de implementación para la función, asegúrese de que sea compatible con la arquitectura del conjunto de instrucciones que elija.
5. Expanda Advanced settings (Configuración avanzada) y, a continuación, seleccione Enable tags (Habilitar etiquetas).
6. Elija Add new tag (Agregar nueva etiqueta) y, a continuación, escriba una clave y el valor opcional. Para añadir más etiquetas, repita este paso.
7. Elija Crear función.

#### Para agregar etiquetas a una función existente

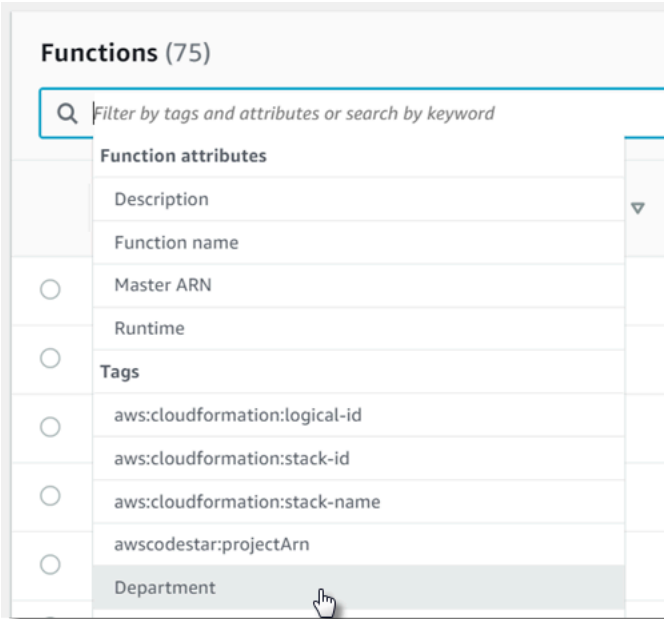
1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija Configuration (Configuración) y, a continuación, elija Tags (Etiquetas).
4. En Etiquetas, elija Administrar etiquetas.
5. Elija Add new tag (Agregar nueva etiqueta) y, a continuación, escriba una clave y el valor opcional. Para añadir más etiquetas, repita este paso.



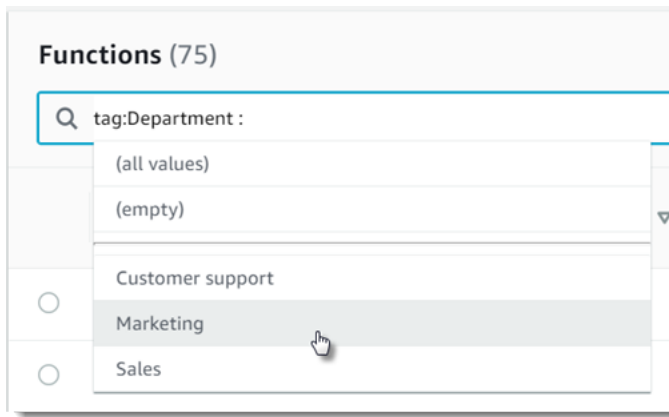
## 6. Seleccione Guardar.

Para filtrar funciones con etiquetas

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija la barra de búsqueda para ver una lista de los atributos de función y las claves de etiqueta.



3. Elija una clave de etiqueta para ver la lista de valores que están en uso en la región de AWS actual.
4. Elija un valor para ver las funciones con ese valor o elija (all values) [(todos los valores)] para ver todas las funciones que tienen una etiqueta con esa clave.



La barra de búsqueda también permite buscar claves de etiqueta. Escriba `tag` para ver de forma exclusiva una lista de claves de etiqueta o escriba el nombre de una clave para encontrarla en la lista.

## Uso de etiquetas con la AWS CLI

### Agregar y eliminar etiquetas

Para crear una nueva función de Lambda con etiquetas, utilice el comando `create-function` con la opción `--tags`.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs20.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

Para añadir etiquetas a una función existente, utilice el comando `tag-resource`.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

Para eliminar etiquetas, utilice el comando `untag-resource`.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:function:my-function \
--tag-keys Department
```



## Visualización de etiquetas de una función

Si desea ver las etiquetas que se aplican a una función de Lambda específica, puede utilizar cualquiera de los siguientes comandos de la AWS CLI:

- [ListTags](#)— Para ver una lista de las etiquetas asociadas a esta función, incluya el ARN (Amazon Resource Name) de su función Lambda:

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:function:my-function
```

- [GetFunction](#)— Para ver una lista de las etiquetas asociadas a esta función, incluya el nombre de la función Lambda:

```
aws lambda get-function --function-name my-function
```

### Filtrado de funciones por etiqueta

Puede utilizar la operación de AWS Resource Groups Tagging API [GetResources](#) API para filtrar los recursos por etiquetas. La operación `GetResources` recibe hasta 10 filtros y cada uno contiene una clave de etiqueta y hasta 10 valores de etiqueta. Usted proporciona `GetResources` con un `ResourceType` para filtrar por tipos de recurso específicos.

Para obtener más información acerca de AWS Resource Groups, consulte [¿Qué son los grupos de recursos?](#) en la Guía del usuario de AWS Resource Groups y etiquetas.

## Requisitos para etiquetas

Los siguientes requisitos se aplican a las etiquetas:

- Número máximo de etiquetas por recurso: 50
- Longitud máxima de la clave: 128 caracteres Unicode en UTF-8
- Longitud máxima del valor: 256 caracteres Unicode en UTF-8
- Las claves y los valores de las etiquetas distinguen entre mayúsculas y minúsculas.
- No utilice el prefijo `aws :` en los nombres o valores de las etiquetas, porque está reservado para uso de AWS. Los nombres y valores de etiquetas que tienen este prefijo no se pueden editar ni eliminar. Las etiquetas que tengan este prefijo no cuentan para el límite de etiquetas por recurso.

- Si pretende utilizar su esquema de etiquetado en múltiples servicios y recursos, recuerde que otros servicios pueden tener restricciones de caracteres permitidos. Los caracteres generalmente permitidos son: letras, espacios y números representables en UTF-8, además de los siguientes caracteres especiales: + - = . \_ : / @.

# Transmisión de respuestas para funciones de Lambda

Puede configurar las URL de función de Lambda para devolver las cargas de respuesta a los clientes. La transmisión de respuestas puede beneficiar a las aplicaciones sensibles a la latencia al mejorar el rendimiento del tiempo hasta el primer byte (TTFB). Esto se debe a que puede volver respuestas parciales al cliente a medida que estén disponibles. Además, puede usar la transmisión de respuestas para crear funciones que devuelvan cargas más grandes. Las cargas de transmisión de respuestas tienen un límite flexible de 20 MB, en comparación con el límite de 6 MB para las respuestas almacenadas en búfer. Transmitir una respuesta también significa que la función no necesita incluir toda la respuesta en la memoria. Para respuestas muy grandes, esto puede reducir la cantidad de memoria que necesita configurar para la función.

La velocidad a la que Lambda transmite las respuestas depende del tamaño de la respuesta. La velocidad de transmisión de los primeros 6 MB de la respuesta de la función no tiene límite. Para las respuestas de más de 6 MB, el resto de la respuesta está sujeto a un límite de ancho de banda. Para obtener más información sobre el ancho de banda de transmisión, consulte [Límites de ancho de banda para la transmisión de respuestas](#).

Las respuestas de transmisión tienen un costo. Para más información, consulte [Precios de AWS Lambda](#).

Lambda admite la transmisión de respuestas en tiempos de ejecución administrados por Node.js. Para otros idiomas, puede [usar un tiempo de ejecución personalizado con una integración de API de tiempo de ejecución personalizada](#) para transmitir respuestas o utilizar [Lambda Web Adapter](#). Puede transmitir las respuestas a través de las [URL de la función de Lambda](#), el AWS SDK o mediante la API de Lambda [InvokeWithResponseStream](#).

## Note

Quando pruebe la función en la consola de Lambda, siempre verá las respuestas en búfer.

## Temas

- [Límites de ancho de banda para la transmisión de respuestas](#)
- [Escritura de funciones de Lambda habilitadas para la transmisión de respuestas](#)
- [Invocación de una función habilitada para la transmisión de respuestas con URL de la función de Lambda](#)
- [Tutorial: Creación de una función de Lambda de transmisión de respuesta con una URL de función](#)

## Límites de ancho de banda para la transmisión de respuestas

Los primeros 6 MB de la carga de respuesta de la función tienen un ancho de banda ilimitado. Tras esta ráfaga inicial, Lambda transmite la respuesta a una velocidad máxima de 2 Mbps. Si las respuestas de la función nunca superan los 6 MB, este límite de ancho de banda nunca se aplica.

### Note

Los límites de ancho de banda solo se aplican a la carga de respuesta de la función y no al acceso de la función a la red.

La velocidad del ancho de banda ilimitado varía en función de diversos factores, incluida la velocidad de procesamiento de la función. Por lo general, puede esperar una velocidad superior a 2 Mbps para los primeros 6 MB de respuesta de la función. Si la función transmite una respuesta a un destino externo a AWS, la velocidad de transmisión también depende de la velocidad de la conexión a Internet externa.

## Escritura de funciones de Lambda habilitadas para la transmisión de respuestas

Escribir el controlador para las funciones de transmisión de respuestas es diferente a escribir los patrones de controlador típicos. Al escribir funciones de transmisión, asegúrese de realizar lo siguiente:

- Ajuste su función con el decorador `awsLambda.streamifyResponse()` que proporcionan los tiempos de ejecución nativos de Node.js.
- Finalice la transmisión de manera correcta para asegurarse de que se haya completado todo el procesamiento de datos.

## Configuración de una función de controlador para transmitir respuestas

Para indicar al tiempo de ejecución que Lambda debe transmitir las respuestas de su función, debe ajustar la función al decorador `streamifyResponse()`. Esto indica al tiempo de ejecución que utilice la ruta lógica adecuada para transmitir las respuestas y permite que la función transmita las respuestas.

El decorador `streamifyResponse()` acepta una función que acepta los siguientes parámetros:

- `event`: proporciona información sobre el evento de invocación de la URL de función, como el método HTTP, los parámetros de consulta y el cuerpo de la solicitud.
- `responseStream`: proporciona una transmisión con escritura permitida.
- `context`: proporciona métodos y propiedades con información acerca de la invocación, la función y el entorno de ejecución.

El objeto `responseStream` es un [writableStream de Node.js](#). Al igual que con cualquier transmisión de este tipo, debe utilizar el método `pipeline()`.

Example controlador habilitado para transmisión de respuestas

```
const pipeline = require("util").promisify(require("stream").pipeline);
const { Readable } = require('stream');

exports.echo = awslambda.streamifyResponse(async (event, responseStream, _context) => {
  // As an example, convert event to a readable stream.
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));

  await pipeline(requestStream, responseStream);
});
```

Si bien `responseStream` ofrece el método `write()` para escribir en la transmisión, le recomendamos que utilice [pipeline\(\)](#) siempre que sea posible. El uso de `pipeline()` garantiza que la transmisión con escritura permitida no se sature por una transmisión legible más rápida.

## Finalización de la transmisión

Asegúrese de finalizar la transmisión correctamente antes de que el controlador regrese. El método `pipeline()` gestiona esto de manera automática.

Para otros casos de uso, llame al método `responseStream.end()` para finalizar correctamente una transmisión. Este método indica que no se deben escribir más datos en la transmisión. Este método no es necesario si escribe a la transmisión con `pipeline()` o `pipe()`.

Example Ejemplo de finalización de una transmisión con `pipeline()`

```
const pipeline = require("util").promisify(require("stream").pipeline);

exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
=> {
```

```
await pipeline(requestStream, responseStream);
});
```

### Example Ejemplo de finalización de una transmisión sin pipeline()

```
exports.handler = awslambda.streamifyResponse(async (event, responseStream, _context)
=> {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

## Invocación de una función habilitada para la transmisión de respuestas con URL de la función de Lambda

### Note

Debe invocar la función mediante una URL de función para transmitir las respuestas.

Puede invocar funciones habilitadas para la transmisión de respuestas si cambia el modo de invocación de la URL de función. El modo de invocación determina qué operación de API utiliza Lambda para invocar la función. Estos son los modos de invocación disponibles:

- **BUFFERED**: esta es la opción predeterminada. Lambda invoca su función mediante la operación de la API `Invoke`. Los resultados de la invocación estarán disponibles cuando se complete la carga. El tamaño de carga máximo es de 6 MB.
- **RESPONSE\_STREAM**: permite que la función transmita los resultados de la carga a medida que estén disponibles. Lambda invoca su función mediante la operación de la API `InvokeWithResponseStream`. El tamaño máximo de carga de respuesta es de 20 MB. Sin embargo, puede [solicitar un aumento de cuota](#).

Aún puede invocar la función sin transmisión de respuestas al llamar directamente a la operación de la API `Invoke`. Sin embargo, Lambda transmite todas las cargas de respuesta para las invocaciones que llegan a través de la URL de función hasta que cambie el modo de invocación a **BUFFERED**.

## Console

Para establecer el modo de invocación de una URL de función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función para la que desea establecer el modo de invocación.
3. Elija la pestaña Configuration (Configuración) y, a continuación, elija Function URL (URL de función).
4. Elija Editar y, a continuación, elija Configuración adicional.
5. En Modo de invocación, elija el modo de invocación que desee.
6. Seleccione Guardar.

## AWS CLI

Para establecer el modo de invocación de la URL de una función (AWS CLI)

```
aws lambda update-function-url-config \  
  --function-name my-function \  
  --invoke-mode RESPONSE_STREAM
```

## AWS CloudFormation

Para establecer el modo de invocación de la URL de una función (AWS CloudFormation)

```
MyFunctionUrl:  
  Type: AWS::Lambda::Url  
  Properties:  
    AuthType: AWS_IAM  
    InvokeMode: RESPONSE_STREAM
```

Para obtener más información acerca de la configuración de las URL de función, consulte las [URL de función de Lambda](#).

## Tutorial: Creación de una función de Lambda de transmisión de respuesta con una URL de función

En este tutorial, se crea una función de Lambda definida como un archivo .zip con un punto de conexión de URL de función que devuelve una transmisión de respuesta. Para obtener más información acerca de la configuración de las URL de funciones, consulte [URL de funciones](#).

### Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#) para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

#### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.



## Creación de un rol de ejecución

Cree el [rol de ejecución](#) que concederá a su función de Lambda permiso para obtener acceso a los recursos de AWS.

Para crear un rol de ejecución

1. Abra la página de [Roles](#) de la consola de AWS Identity and Access Management (IAM).
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes:
  - Tipo de entidad de confianza: servicio de AWS
  - Caso de uso: Lambda
  - Permisos: AWSLambdaBasicExecutionRole
  - Role name (Nombre de rol): **response-streaming-role**

La política AWSLambdaBasicExecutionRole tiene permisos que la función necesita para escribir registros a Registros de Amazon CloudWatch. Después de crear el rol, anote su nombre de recurso de Amazon (ARN). Lo necesitará en el siguiente paso.

## Crear una función de transmisión de respuestas (AWS CLI)

Cree una función de Lambda de transmisión de respuesta con un punto de conexión de URL de función mediante la AWS Command Line Interface (AWS CLI).

Para crear una función que pueda transmitir respuestas

1. Copie el siguiente código de ejemplo en un archivo denominado `index.mjs`.

```
import util from 'util';
import stream from 'stream';
const { Readable } = stream;
const pipeline = util.promisify(stream.pipeline);

/* global awslambda */
export const handler = awslambda.streamifyResponse(async (event, responseStream,
  _context) => {
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));
  await pipeline(requestStream, responseStream);
});
```

```
});
```

2. Cree un paquete de implementación.

```
zip function.zip index.mjs
```

3. Cree una función de Lambda con el comando `create-function`. Reemplace el valor de `--role` por el ARN del rol del paso anterior.

```
aws lambda create-function \  
  --function-name my-streaming-function \  
  --runtime nodejs16.x \  
  --zip-file fileb://function.zip \  
  --handler index.handler \  
  --role arn:aws:iam::123456789012:role/response-streaming-role
```

Para crear una URL de función

1. Agregue una política basada en recursos a la función para habilitar el acceso a la URL de función. Reemplace el valor de `--principal` por su ID de Cuenta de AWS.

```
aws lambda add-permission \  
  --function-name my-streaming-function \  
  --action lambda:InvokeFunctionUrl \  
  --statement-id 12345 \  
  --principal 123456789012 \  
  --function-url-auth-type AWS_IAM \  
  --statement-id url
```

2. Cree un punto de conexión de la URL para la función con el comando `create-function-url-config`.

```
aws lambda create-function-url-config \  
  --function-name my-streaming-function \  
  --auth-type AWS_IAM \  
  --invoke-mode RESPONSE_STREAM
```

## Prueba del punto de conexión de la URL de función

Invoque la función para probar la integración. Puede abrir la URL de función en un navegador o puede usar curl.

```
curl --request GET "<function_url>" --user "<key:token>" --aws-sigv4 "aws:amz:us-east-1:lambda" --no-buffer
```

Nuestra URL de función utiliza el tipo de autenticación IAM\_AUTH. Esto significa que debe firmar las solicitudes con las clave de acceso y clave secreta de AWS. En el comando anterior, reemplace <key:token> por el ID de clave de acceso de AWS. Ingrese su clave secreta de AWS cuando se le solicite. Si no tiene su clave secreta de AWS, puede [utilizar credenciales de AWS temporales](#) en su lugar.

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y seleccione Delete (Eliminar).

# Comprender los métodos de invocación de la función de Lambda

Después de implementar la función de Lambda, puede invocarla de varias maneras:

- La [consola de Lambda](#): utilice la consola de Lambda para crear rápidamente un evento de prueba a fin de invocar la función.
- [AWS SDK](#): utilice AWS SDK para invocar la función mediante programación.
- API [Invoke](#): utilice la API Invoke de Lambda para invocar directamente la función.
- La [AWS Command Line Interface \(AWS CLI\)](#): utilice el comando `aws lambda invoke` de la AWS CLI para invocar directamente la función desde la línea de comandos.
- Un [punto de conexión HTTP\(S\)](#): utilice las URL de función para crear un punto de conexión HTTP(S) dedicado que pueda utilizar para invocar la función.

Todos estos métodos son formas directas de invocar la función. En Lambda, un caso de uso común es invocar la función según un evento que se produce en otra parte de la aplicación. Algunos servicios pueden invocar una función de Lambda con cada nuevo evento. Esto se llama [desencadenador](#). Para los servicios basados en flujos y colas, Lambda invoca la función con lotes de registros. Esto se denomina [asignación de orígenes de eventos](#).

Al invocar una función, puede optar por invocarla de forma síncrona o asíncrona. Con [invocación síncrona](#), espere la función para procesar el evento y devolver una respuesta. Con invocación [asíncrona](#), Lambda pone en cola el evento para su procesamiento y devuelve una respuesta inmediatamente. El [parámetro de solicitud `InvocationType` de la API Invoke](#) determina cómo Lambda invoca la función. Un valor de `RequestResponse` indica una invocación síncrona y un valor de `Event` indica una invocación asíncrona.

[Para invocar su función a través de IPv6, utilice los puntos finales públicos de doble pila de Lambda.](#)

Puntos de conexión de doble pila compatibles con IPv4 e IPv6 Los puntos de conexión de doble pila de Lambda utilizan la siguiente sintaxis:

```
protocol://lambda.us-east-1.api.aws
```

También puede usar las [URL de función de Lambda](#) para invocar funciones a través de IPv6. Los puntos de conexión de la URL de función tienen el siguiente formato:

```
https://url-id.lambda-url.us-east-1.on.aws
```

Si la invocación de la función produce un error, en el caso de las invocaciones sincrónicas, consulte el mensaje de error en la respuesta y vuelva a intentar la invocación manualmente. Para invocaciones asíncronas, Lambda gestiona los reintentos automáticamente y puede enviar registros de invocación a un [destino](#).

## Invocación de una función de Lambda de forma sincrónica

Al invocar una función sincrónicamente, Lambda ejecuta la función y espera una respuesta. Cuando finaliza la función, Lambda devuelve la respuesta desde el código de la función con datos adicionales, como la versión de la función que se invocó. Para invocar una función de forma síncrona con la AWS CLI, utilice el comando `invoke`.

```
aws lambda invoke --function-name my-function \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "key": "value" }' response.json
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

El siguiente diagrama muestra a los clientes que invocan una función de Lambda de forma sincrónica. Lambda envía los eventos directamente a la función y envía la respuesta de la función al invocador.



El `payload` es una cadena que contiene un evento en formato JSON. El nombre del archivo donde AWS CLI escribe la respuesta de la función es `response.json`. Si la función devuelve un objeto o error, el cuerpo de la respuesta es el objeto o error en formato JSON. Si la función sale sin errores, el cuerpo de la respuesta es `null`.

### Note

Lambda no espera a que se completen las extensiones externas para enviar la respuesta. Las extensiones se ejecutan como un proceso independiente en el entorno de ejecución y puede continuar ejecutándose luego de que la invocación de la función se complete. Para obtener más información, consulte [Aumente las funciones de Lambda utilizando extensiones de Lambda](#).

La salida del comando, que se muestra en el terminal, incluye información de los encabezados en la respuesta de Lambda. Esto incluye la versión que ha procesado el evento (útil cuando se utilizan [alias](#)) y el código de estado devuelto por Lambda. Si Lambda pudo ejecutar la función, el código de estado es 200, incluso aunque la función devolviera un error.

### Note

Para funciones con un tiempo de espera largo, el cliente puede desconectarse durante la invocación síncrona mientras espera una respuesta. Configure el cliente HTTP, SDK, el firewall, el proxy o el sistema operativo para permitir conexiones largas con tiempo de espera o ajustes `keep-alive`.

Si Lambda no puede ejecutar la función, el error se muestra en la salida.

```
aws lambda invoke --function-name my-function \  
  --cli-binary-format raw-in-base64-out \  
  --payload value response.json
```

Debería ver los siguientes datos de salida:

```
An error occurred (InvalidRequestContentException) when calling the Invoke operation:  
Could not parse request body into json: Unrecognized token 'value': was expecting  
( 'true', 'false' or 'null' )
```

```
at [Source: (byte[])"value"; line: 1, column: 11]
```

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBU1QgUmVxdWVzdElk0iA4N2QwNDRi0C1mMTU0LTExZTgt0GNkYS0y0Tc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-`



in-base64-out. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad base64 está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar base64 -D.

Para obtener más información acerca de la API Invoke, incluida una lista completa de parámetros, encabezados y errores, consulte [Invocar](#).

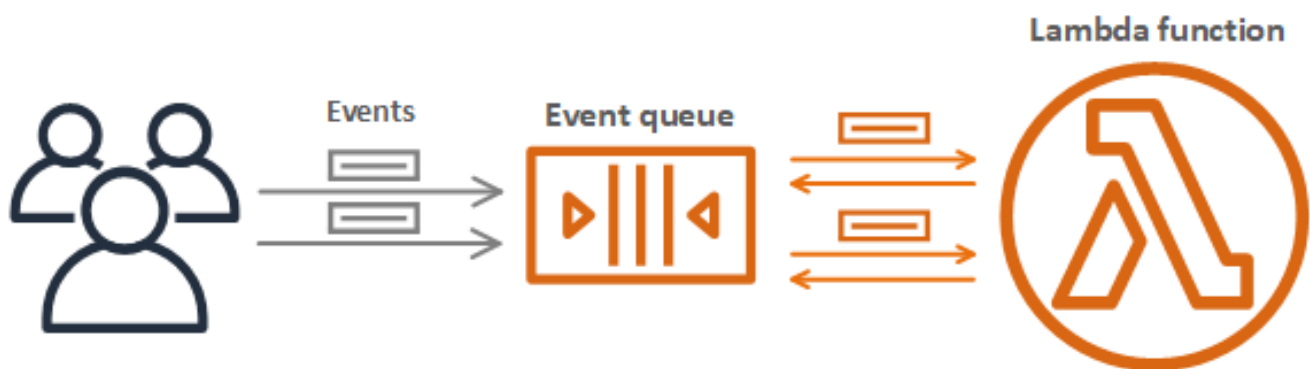
Al invocar una función directamente, puede comprobar la respuesta en busca de errores y volver a intentarlo. La AWS CLI y el SDK de AWS también reintentan automáticamente en los tiempos de espera, limitaciones y errores de servicio del cliente. Para obtener más información, consulte [Comprender el comportamiento de reintento en Lambda](#).

## Cómo Lambda administra las invocaciones asíncronas

Varios Servicios de AWS, como de Amazon Simple Storage Service (Amazon S3) y Amazon Simple Notification Service (Amazon SNS), invocan funciones de forma asíncrona para procesar eventos. También puede invocar una función de Lambda de forma asíncrona mediante la AWS Command Line Interface (AWS CLI) o los AWS SDK. Cuando se invoca una función de forma asíncrona, no se espera una respuesta del código de función. Se entrega el evento a Lambda, y Lambda se ocupa del resto. Puede configurar la forma en que Lambda gestiona los errores y enviar registros de invocaciones a un recurso posterior, como Amazon Simple Queue Service (Amazon SQS) o Amazon EventBridge (EventBridge), para encadenar los componentes de la aplicación.

El siguiente diagrama muestra los clientes que invocan una función de Lambda de forma asíncrona. Lambda pone en cola los eventos antes de enviarlos a la función.

### Asynchronous Invocation



Para la invocación asíncrona, Lambda coloca el evento en una cola y devuelve una respuesta de “proceso realizado con éxito” sin información adicional. Un proceso independiente lee eventos de la cola y los envía a la función.

### Invocación de una función de Lambda de forma asíncrona

Para invocar una función de Lambda de forma asíncrona mediante la AWS Command Line Interface (AWS CLI) o los AWS SDK, establezca el parámetro de tipo de invocación en Event. En el siguiente ejemplo se muestra un comando de la AWS CLI para invocar una función.

```
aws lambda invoke \  
  --function-name my-function \  
  \
```

```
--invocation-type Event \  
--cli-binary-format raw-in-base64-out \  
--payload '{ "key": "value" }' response.json
```

Debería ver los siguientes datos de salida:

```
{  
  "StatusCode": 202  
}
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

El archivo de salida (`response.json`) no contiene ninguna información, pero se crea al ejecutar este comando. Si Lambda no puede añadir el caso a la cola, el mensaje de error aparece en la salida del comando.

## Cómo Lambda administra los errores y los reintentos mediante la invocación asíncrona

Lambda administra la cola de eventos asíncrona de la función y vuelve a intentarlo en caso de errores. Si la función devuelve un error, de forma predeterminada, Lambda intenta ejecutarla dos veces más, con una espera de un minuto entre los dos primeros intentos y dos minutos entre el segundo y el tercero. Los errores de la función incluyen errores devueltos por el código de la función y los errores devueltos por el tiempo de ejecución de la función, como, por ejemplo, los tiempos de espera.

Si la función no tiene disponible suficiente simultaneidad para procesar todos los eventos, se limitan las solicitudes adicionales. Para la limitación controlada de errores (429) y errores del sistema (serie 500), de forma predeterminada, Lambda devuelve el evento a la cola e intenta ejecutar la función de nuevo durante un máximo de 6 horas. El intervalo de reintento aumenta exponencialmente desde 1 segundo después del primer intento hasta un máximo de 5 minutos. Si la cola contiene muchas entradas, Lambda aumenta el intervalo de reintento y reduce la velocidad a la que lee eventos de la cola.

Aunque la función no devuelva un error, es posible que reciba el mismo evento de Lambda varias veces, ya que la propia cola ofrece consistencia final. Si la función no es capaz de gestionar los eventos entrantes, podrían también eliminarse de la cola sin que se envíen a la función. Asegúrese de que el código de la función gestione sin problemas eventos duplicados y de que tenga simultaneidad suficiente disponible para gestionar todas las invocaciones.

Cuando la cola es muy larga, es posible que los nuevos eventos se agoten antes de que Lambda pueda enviarlos a la función. Cuando un evento caduca o todos los intentos de procesamiento fallan, Lambda lo descarga. Puede [configurar la administración de errores](#) de una función para reducir el número de reintentos que realiza Lambda o para descartar eventos no procesados más rápidamente.

También puede configurar Lambda para que envíe un registro de invocación a otro servicio. Consulte [Captura de registros de invocaciones asíncronas de Lambda](#) para obtener más información.

## Configuración de la invocación asíncrona de Lambda mediante la API

Para administrar la configuración de la invocación asíncrona con la AWS CLI o el AWS SDK, utilice las siguientes operaciones de la API:

- [PutFunctionEventInvokeConfig](#)
- [GetFunctionEventInvokeConfig](#)
- [UpdateFunctionEventInvokeConfig](#)
- [ListFunctionEventInvokeConfigs](#)
- [DeleteFunctionEventInvokeConfig](#)

## Configuración de la invocación asíncrona (AWS CLI)

Para configurar la invocación asíncrona con la CLI de AWS CLI, utilice el comando `put-function-event-invoke-config`. En el ejemplo siguiente se configura una función con una antigüedad máxima de evento de 1 hora y sin reintentos.

```
aws lambda put-function-event-invoke-config --function-name error \  
--maximum-event-age-in-seconds 3600 --maximum-retry-attempts 0
```

Debería ver los siguientes datos de salida:

```
{  
  "LastModified": 1573686021.479,  
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
```

```

    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
      "OnSuccess": {},
      "OnFailure": {}
    }
  }
}

```

El comando `put-function-event-invoke-config` sobrescribe cualquier configuración existente en la función, versión o alias. Para configurar una opción sin reiniciar otras, utilice `update-function-event-invoke-config`. En el siguiente ejemplo, se configura Lambda para enviar un registro a una cola de SQS estándar llamada `destination` cuando no se puede procesar un evento.

```

aws lambda update-function-event-invoke-config --function-name error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'

```

Debería ver los siguientes datos de salida:

```

{
  "LastModified": 1573687896.493,
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
  "MaximumRetryAttempts": 0,
  "MaximumEventAgeInSeconds": 3600,
  "DestinationConfig": {
    "OnSuccess": {},
    "OnFailure": {
      "Destination": "arn:aws:sqs:us-east-2:123456789012:destination"
    }
  }
}

```

## Configuración de la administración de errores para la invocación asíncrona (consola)

Utilice la consola de Lambda para configurar los ajustes de gestión de errores en una función, una versión o un alias.

Para configurar la gestión de errores

1. Abra la [página de Funciones](#) en la consola de Lambda.

2. Elija una función.
3. Elija Configuración y, a continuación, elija Invocación asíncrona.
4. En Asynchronous invocation (Invocación asincrónica), elija Edit (Editar).
5. Configure los siguientes ajustes.
  - Antigüedad máxima del evento: el período máximo de tiempo durante el que Lambda retiene un evento en la cola de evento asincrónico, hasta 6 horas.
  - Número de reintentos: número de reintentos que Lambda realiza cuando la función devuelve un error, entre 0 y 2.
6. Seleccione Guardar.

Cuando un evento de invocación supera la antigüedad máxima o no supera ningún reintento, Lambda lo descarta. Para conservar una copia de eventos descartados, configure un destino de eventos fallidos.

## Captura de los registros de invocaciones asíncronas de Lambda

Lambda puede enviar registros de invocaciones asíncronas a uno de los siguientes Servicios de AWS.

- Amazon SQS: una cola de SQS estándar.
- Amazon SNS: un tema de SNS estándar.
- AWS Lambda: una función Lambda.
- Amazon EventBridge: el ARN de un bus de eventos de EventBridge.

El registro de invocación contiene detalles sobre la solicitud y la respuesta en formato JSON. Puede configurar destinos independientes en eventos que se procesan con éxito, y eventos que fallan todos los intentos de procesamiento. También puede configurar una cola de Amazon SQS estándar o un tema de Amazon SNS estándar como una [cola de mensajes fallidos](#) para eventos descartados. En las colas de mensajes fallidos, Lambda solo envía el contenido del evento, sin detalles sobre la respuesta.

Si Lambda no puede enviar un registro a un destino que haya configurado, envía una métrica `DestinationDeliveryFailures` a Amazon CloudWatch. Esto puede ocurrir si la configuración incluye un tipo de destino no admitido, como una cola FIFO de Amazon SQS o un tema FIFO de Amazon SNS. También pueden producirse errores de entrega debido a errores de permisos y límites

de tamaño. Para obtener más información sobre las métricas de invocación de Lambda, consulte [Métricas de invocación](#).

#### Note

Para evitar que una función se active, puede establecer la simultaneidad reservada de la función en cero. Cuando establece la simultaneidad reservada en cero para una función invocada de forma asíncrona, Lambda comienza a enviar nuevos eventos a la [cola de mensajes fallidos](#) configurada o al [destino para eventos](#) en caso de error, sin reintentos. Para procesar eventos que se enviaron mientras la simultaneidad reservada estaba establecida en cero, debe consumir los eventos de la cola de mensajes fallidos o el destino para eventos en caso de error.

## Configuración de destinos para invocación asíncrona

Para retener registros de invocaciones asíncronas, agregue un destino a su función. Puede elegir enviar las invocaciones correctas o fallidas a un destino. Cada función puede tener varios destinos, por lo que puede configurar destinos independientes para los eventos correctos y los fallidos. Cada registro enviado al destino es un documento JSON con detalles sobre la invocación. Al igual que con los ajustes de gestión de errores, puede configurar los destinos en una función, versión de la función o alias.

#### Note

También puede retener registros de las invocaciones fallidas para los siguientes tipos de asignación de orígenes de eventos: [Amazon Kinesis](#), [Amazon DynamoDB](#), [Apache Kafka autogestionado](#) y [Amazon MSK](#).

La siguiente table enumera los destinos admitidos para los registros de invocación asíncrona. Para que Lambda envíe correctamente los registros al destino elegido, asegúrese de que el [rol de ejecución](#) de la función también contenga los permisos pertinentes. En la tabla también se describe cómo cada tipo de destino recibe el registro de invocación de JSON.

Tipo de destino	Permiso necesario	Formato JSON específico del destino
Cola de Amazon SQS	<a href="#">sqs:SendMessage</a>	Lambda pasa el registro de invocación como Message al destino.
Tema de Amazon SNS	<a href="#">sns:Publish</a>	Lambda pasa el registro de invocación como Message al destino.
Función de Lambda	<a href="#">InvokeFunction</a>	Lambda pasa el registro de invocación como carga útil a la función.
EventBridge	<a href="#">events:PutEvents</a>	<ul style="list-style-type: none"> <li>Lambda pasa el registro de invocación como <code>detail</code> en la llamada <code>PutEvents</code>.</li> <li>El valor del campo de eventos <code>source</code> es <code>lambda</code>.</li> <li>El valor del campo de eventos <code>detail-type</code> es “Resultado de la invocación de la función Lambda: éxito” o “Resultado de invocación de función de Lambda: error”.</li> <li>El campo de eventos <code>resource</code> contiene la función y el destino de Amazon Resource Names (ARN).</li> <li>Para ver otros campos de eventos, consulte <a href="#">Eventos de Amazon EventBridge</a>.</li> </ul>



El ejemplo siguiente muestra un registro de invocación para un evento que ha fallado tres intentos de procesamiento debido a un error de función. El registro de invocación contiene detalles sobre el evento, la respuesta y el motivo por el que se ha enviado el registro.

```
{
  "version": "1.0",
  "timestamp": "2019-11-14T18:16:05.568Z",
  "requestContext": {
    "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:
$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "requestPayload": {
    "ORDER_IDS": [
      "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",
      "637de236-e7b2-464e-xmpl-baf57f86bb53",
      "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"
    ]
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "responsePayload": {
    "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited
before completing request"
  }
}
```

En los siguientes pasos se describe cómo configurar un destino para una función utilizando la consola de Lambda.

Configuración de un destino para registros de invocación asíncrona.

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En Descripción general de la función, elija Agregar destino.
4. En Source (Origen), elija Asynchronous invocation (Invocación asincrónica).

5. En Condition (Condición) elija una de las siguientes opciones:
  - En caso de error: enviar un registro cuando el evento no supera los intentos de procesamiento o supera la antigüedad máxima.
  - Si es correcto: enviar un registro cuando la función procesa correctamente una invocación asíncrona.
6. En Destination type (Tipo de destino), elija el tipo de recurso que recibe el registro de invocación.
7. En Destination (Destino), elija un recurso.
8. Seleccione Guardar.

Cuando una invocación coincide con la condición, Lambda envía un documento JSON con detalles sobre la invocación al destino.

#### Formato JSON específico del destino

- Para Amazon SQS y Amazon SNS (`SnsDestination` y `SqsDestination`), el registro de invocación se pasa como `Message` al destino.
- Para Lambda (`LambdaDestination`), el registro de invocación se pasa como carga útil a la función.
- Para EventBridge (`EventBridgeDestination`), el registro de invocación se pasa como `detail` en la llamada [PutEvents](#). El valor del campo de eventos `source` es `lambda`. El valor del campo de eventos `detail-type` es `Resultado de la invocación de la función Lambda: éxito` o `Resultado de invocación de función Lambda: error`. El campo de eventos `resource` contiene la función y el destino de Amazon Resource Names (ARN). Para ver otros campos de eventos, consulte [Eventos de Amazon EventBridge](#).

El ejemplo siguiente muestra un registro de invocación para un evento que ha fallado tres intentos de procesamiento debido a un error de función.

#### Example Registro de invocación

```
{
  "version": "1.0",
  "timestamp": "2019-11-14T18:16:05.568Z",
  "requestContext": {
    "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
```

```
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:
$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "requestPayload": {
    "ORDER_IDS": [
      "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",
      "637de236-e7b2-464e-xmpl-baf57f86bb53",
      "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"
    ]
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "responsePayload": {
    "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited
before completing request"
  }
}
```

El registro de invocación contiene detalles sobre el evento, la respuesta y el motivo por el que se ha enviado el registro.

## Seguimiento de solicitudes a destinos

Puede utilizar AWS X-Ray para ver una vista conectada de cada solicitud a medida que se pone en cola, la procesa una función de Lambda y se pasa al servicio de destino. Al activar el seguimiento de X-Ray para una función o un servicio que invoca una función, Lambda agrega un encabezado de X-Ray a la solicitud y lo pasa al servicio de destino. El seguimiento de los servicios anteriores se vincula automáticamente al seguimiento de las funciones de Lambda posteriores, lo que crea una vista integral de toda la aplicación. Para obtener más información sobre el seguimiento, consulte [Visualice las invocaciones de la función de Lambda mediante AWS X-Ray](#).

## Colas de mensajes fallidos

Como alternativa a un [destino en caso de fallo](#), puede configurar su función con una cola de mensajes fallidos para guardar eventos descartados para su posterior procesamiento. Una cola de mensajes fallidos actúa igual que un destino en caso de error, ya que se utiliza cuando un evento falla todos los intentos de procesamiento o caduca sin ser procesado. Sin embargo, solo puede

agregar o eliminar una cola de mensajes fallidos a nivel de función. Las versiones de la función usan la misma configuración de cola de mensajes fallidos que la versión no publicada (\$LATEST). Los destinos en caso de error también admiten destinos adicionales e incluyen detalles sobre la respuesta de la función en el registro de invocación.

Para volver a procesar eventos en una cola de mensajes fallidos, puede configurarla como una fuente de eventos para su función de Lambda. También puede recuperar manualmente los eventos.

Puede elegir una cola de Amazon SQS estándar o un tema estándar de Amazon SNS para la cola de mensajes fallidos. No se admiten las colas FIFO ni los temas FIFO de Amazon SNS. Si no se dispone de una cola o tema, debe crearse. Elija el tipo de destino que coincida con su caso de uso.

- [Cola de Amazon SQS](#): una cola que contiene eventos fallidos hasta que se recuperan. Elija una cola estándar de Amazon SQS si espera que una sola entidad, como una función de Lambda o una alarma de CloudWatch, procese el evento fallido. Para obtener más información, consulte [Uso de Lambda con Amazon SQS](#).

Cree una cola en la [consola de Amazon SQS](#).

- [Tema de Amazon SNS](#): un tema transmite eventos fallidos a uno o más destinos. Elija un tema estándar de Amazon SNS si espera que varias entidades actúen en un evento fallido. Por ejemplo, puede configurar un tema para enviar eventos a una dirección de correo electrónico, una función de Lambda o un punto de enlace HTTP. Para obtener más información, consulte [Invocar las funciones de Lambda usando las notificaciones de Amazon SNS](#).

Cree un tema en la [consola de Amazon SNS](#).

Para enviar eventos a una cola o tema, la función necesita permisos adicionales. Añada una política con los permisos necesarios al [rol de ejecución](#) de la función.

- Amazon SQS: [sqs:SendMessage](#)
- Amazon SNS: [sns:Publish](#)

Si la cola o el tema de destino están cifrados con una clave administrada por el cliente, el rol de ejecución también debe ser un usuario en la [política basada en recursos](#) de la clave.

Después de crear el destino y actualizar el rol de ejecución de la función, añada la cola de mensajes fallidos a la función. Puede configurar varias funciones para enviar eventos al mismo destino.

## Para configurar una cola de mensajes fallidos

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuración y, a continuación, elija Invocación asíncrona.
4. En Asynchronous invocation (Invocación asincrónica), elija Edit (Editar).
5. Establezca el Recurso de DLQ en Amazon SQS o Amazon SNS.
6. Elija la cola o el tema de destino.
7. Seleccione Guardar.

Para configurar una cola de mensajes fallidos con la AWS CLI, utilice el comando `update-function-configuration`.

```
aws lambda update-function-configuration --function-name my-function \  
--dead-letter-config TargetArn=arn:aws:sns:us-east-2:123456789012:my-topic
```

Lambda envía el evento a la cola de mensajes fallidos tal y como está, con información adicional en atributos. Puede utilizar esta información para identificar el error que la función devuelve o correlacionar el evento con los registros o un rastro de AWS X-Ray.

### Atributos de mensajes de cola de mensajes fallidos

- RequestID (cadena): el ID de la solicitud de invocación. Los ID de las solicitudes aparecen en los registros de la función. También puede usar el X-Ray SDK para registrar el ID de solicitud en un atributo del rastro. A continuación, puede buscar rastros por ID de solicitud en la consola de X-Ray.
- ErrorCode (número): el código de estado de HTTP.
- ErrorMessage (cadena): el primer 1 KB del mensaje de error.

Si Lambda no puede enviar un mensaje a la cola de mensajes fallidos, elimina el evento y emite la métrica [DeadLetterErrors](#). Esto puede ocurrir debido a la falta de permisos o si el tamaño total del mensaje supera el límite de la cola o tema de destino. Por ejemplo, supongamos que una notificación de Amazon SNS con un cuerpo cercano a los 256 KB activa una función que genera un error. En ese caso, los datos del evento que Amazon SNS agrega, junto con los atributos agregados por Lambda, pueden hacer que el mensaje supere el tamaño máximo permitido en la cola de mensajes fallidos.

---

Si está utilizando Amazon SQS como fuente de eventos, configure una cola de mensajes fallidos en la propia cola de Amazon SQS y no en la función de Lambda. Para obtener más información, consulte [Uso de Lambda con Amazon SQS](#).

# Cómo procesa Lambda registros de orígenes de eventos basados en secuencias y colas

Una asignación de orígenes de eventos es un recurso de Lambda que lee elementos de servicios basados en secuencias y colas e invoca una función con lotes de registros. Los siguientes servicios utilizan asignaciones de orígenes de eventos para invocar las funciones de Lambda:

- [Amazon DocumentDB \(con compatibilidad con MongoDB\) \(Amazon DocumentDB\)](#)
- [Amazon DynamoDB](#)
- [Amazon Kinesis](#)
- [Amazon MQ](#)
- [Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#)
- [Apache Kafka autoadministrado](#)
- [Amazon Simple Queue Service \(Amazon SQS\)](#)

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

## En qué se diferencian las asignaciones de orígenes de eventos de los desencadenadores directos

Algunos servicios de AWS pueden invocar directamente las funciones de Lambda mediante desencadenadores. Estos servicios envían eventos a Lambda y la función se invoca inmediatamente cuando se produce el evento especificado. Los desencadenadores son adecuados para eventos discretos y para el procesamiento en tiempo real. Al [crear un desencadenador mediante la consola de Lambda](#), esta interactúa con el servicio de AWS correspondiente para configurar la notificación de eventos en ese servicio. En realidad, el servicio que genera los eventos es el que almacena y

administra el desencadenador, no Lambda. Estos son algunos ejemplos de servicios que utilizan desencadenadores para invocar funciones de Lambda:

- Amazon Simple Storage Service (Amazon S3): invoca una función cuando se crea, elimina o modifica un objeto en un bucket. Para obtener más información, consulte [Tutorial: Uso de un desencadenador de Amazon S3 para invocar una función de Lambda](#).
- Amazon Simple Notification Service (Amazon SNS): invoca una función cuando se publica un mensaje en un tema de SNS. Para obtener más información, consulte [Tutorial: Uso de AWS Lambda con Amazon Simple Notification Service](#).
- Amazon API Gateway: invoca una función cuando se realiza una solicitud de la API a un punto de conexión específico. Para obtener más información, consulte [Invocación de una función de Lambda mediante un punto de conexión de Amazon API Gateway](#).

Las asignaciones de orígenes de eventos son recursos de Lambda creados y administrados dentro del servicio Lambda. Las asignaciones de orígenes de eventos están diseñados para procesar grandes volúmenes de datos o mensajes de streaming procedentes de colas. Procesar los registros de una secuencia o una cola por lotes es más eficiente que procesar los registros de forma individual.

## Comportamiento de procesamiento por lotes


De forma predeterminada, una asignación de origen de eventos agrupa los registros en una sola carga que Lambda envía a su función. Para ajustar el comportamiento de procesamiento por lotes, puede configurar una ventana de procesamiento por lotes ([MaximumBatchingWindowInSeconds](#)) y un tamaño del lote ([BatchSize](#)). Un periodo de procesamiento por lotes es la cantidad de tiempo máxima para recopilar registros en una sola carga. El tamaño del lote es el número máximo de registros de un solo lote. Lambda invoca su función cuando se cumple uno de los tres criterios siguientes:

- El plazo de procesamiento por lotes alcanza su valor máximo. El comportamiento predeterminado del plazo de procesamiento por lotes varía en función del origen de eventos específico.
- Para los orígenes de eventos de Kinesis, DynamoDB y Amazon SQS: el plazo de procesamiento por lotes predeterminado es de 0 segundos. Esto significa que Lambda invoca su función tan pronto como los registros estén disponibles. Para establecer un plazo de procesamiento por lotes, configure `MaximumBatchingWindowInSeconds`. Puede establecer este parámetro en cualquier valor entre 0 y 300 segundos, en incrementos de 1 segundo. Si configura un plazo de



procesamiento por lotes, el siguiente plazo comienza tan pronto como se completa la invocación de la función anterior.

- En el caso de los orígenes de eventos de Amazon MSK, Apache Kafka autoadministrado, Amazon MQ y Amazon DocumentDB: el periodo de procesamiento por lotes predeterminado es de 500 ms. Puede configurar `MaximumBatchingWindowInSeconds` como cualquier valor entre 0 segundos y 300 segundos, en incrementos de segundos. Un plazo de procesamiento por lotes comienza en cuanto llega el primer registro.

 Note

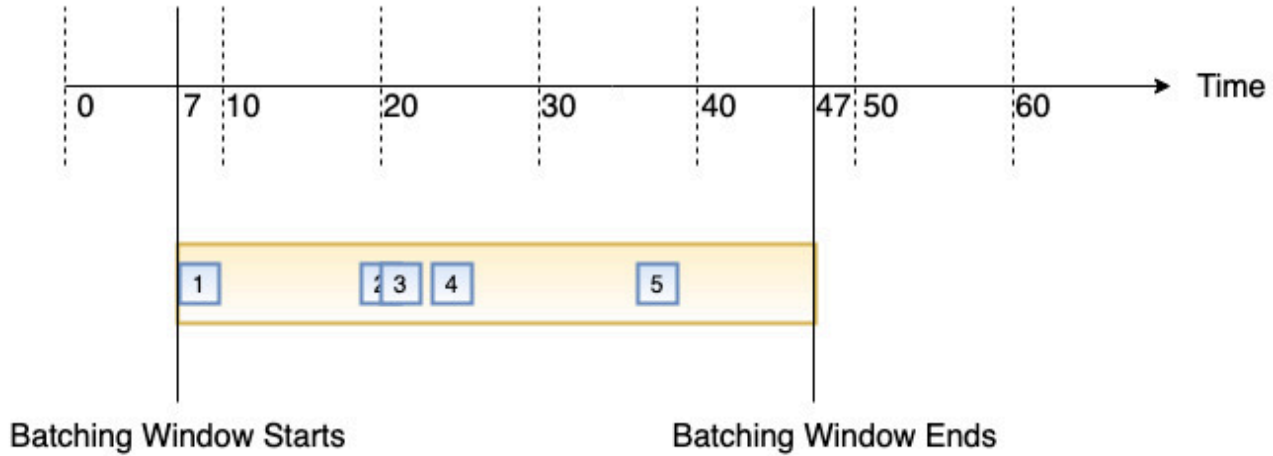
Como solo puede cambiar `MaximumBatchingWindowInSeconds` en incrementos de segundos, no puede volver al plazo de procesamiento por lotes predeterminado de 500 ms después de haberlo cambiado. Para restaurar el plazo de procesamiento por lotes predeterminado, debe crear una nueva asignación de origen de eventos.

- Se cumple el tamaño del lote. El tamaño mínimo del lote es 1. El tamaño predeterminado y máximo del lote depende del origen de eventos. Para obtener más información sobre estos valores, consulte la especificación [BatchSize](#) para la operación de la API de `CreateEventSourceMapping`.
- El tamaño de la carga alcanza los [6 MB](#). Este límite no se puede modificar.

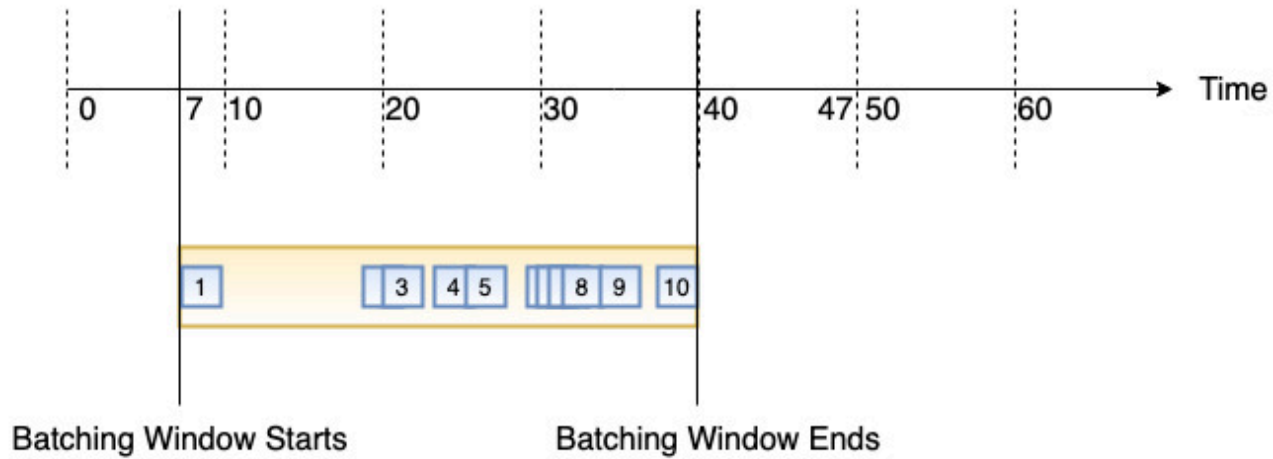
En el siguiente diagrama se ilustran estas tres condiciones. Supongamos que un plazo de procesamiento por lotes comienza a los  $t = 7$  segundos. En el primer escenario, el plazo de procesamiento por lotes alcanza su máximo de 40 segundos a los  $t = 47$  segundos después de acumular 5 registros. En el segundo escenario, el tamaño del lote llega a 10 antes de que venza el plazo de procesamiento por lotes, por lo que el plazo de procesamiento por lotes finaliza antes de tiempo. En el tercer escenario, se alcanza el tamaño máximo de la carga antes de que venza el plazo de procesamiento por lotes, por lo que el plazo de procesamiento por lotes finaliza antes de tiempo.

Max Batching Window = 40 Seconds  
Max Batch Size = 10  
Max Batch Size in Bytes = 6 MB

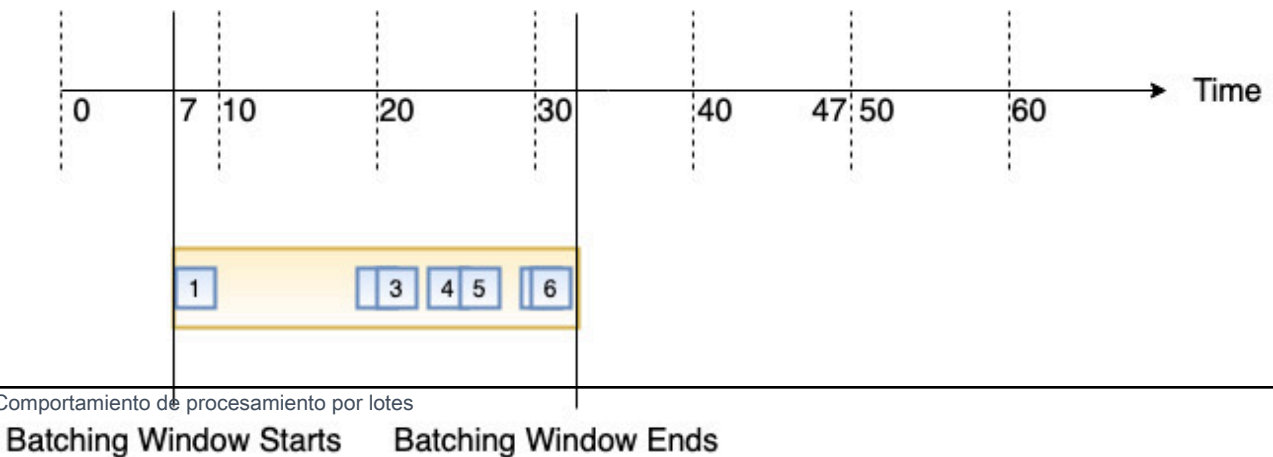
(1) Batching Window Expires



(2) Batching Size is reached



(3) Batch Size in bytes is reached



Recomendamos que pruebe con diferentes tamaños de lote y de registro para que la frecuencia de sondeo de cada origen de eventos se ajuste a la velocidad con la que la función es capaz de completar su tarea. El parámetro de BatchSize [CreateEventSourceMapping](#) controla el número máximo de registros que se pueden enviar a la función en cada invocación. A menudo, un tamaño de lote mayor puede absorber de forma más eficiente el tráfico adicional asociado a un conjunto de registros mayor, mejorando el desempeño.

Lambda no espera a que se complete una [extensión](#) configurada antes de enviar el siguiente lote para su procesamiento. En otras palabras, las extensiones pueden seguir ejecutándose mientras Lambda procesa el siguiente lote de registros. Esto puede provocar problemas de limitación si infringe alguno de los ajustes o límites de [simultaneidad](#) de la cuenta. Para detectar si se trata de un posible problema, supervise sus funciones y compruebe si ve [métricas de simultaneidad](#) más elevadas de lo esperado para la asignación de orígenes de eventos. Debido a los tiempos cortos entre invocaciones, Lambda puede informar brevemente un uso de simultaneidad superior al número de particiones. Esto puede ser cierto incluso para las funciones de Lambda sin extensiones.

De forma predeterminada, si su función devuelve un error, la asignación de origen de eventos vuelve a procesar todo el lote hasta que la función se complete correctamente o los elementos del lote venzan. Para garantizar el procesamiento en orden, la asignación de origen de eventos mantiene en pausa el procesamiento de la partición afectada hasta que se resuelve el error. Para los orígenes de flujos (DynamoDB y Kinesis), puede configurar la cantidad máxima de reintentos que Lambda puede realizar cuando la función devuelva un error. Los errores del servicio o las limitaciones que se producen cuando el lote no llega a la función no se tienen en cuenta para la cantidad de reintentos. También puede configurar la asignación de orígenes de eventos para enviar un registro de invocación a un [destino](#) cuando descarta un lote de eventos.

## API de asignación de orígenes de eventos

Para administrar un origen de eventos con la [AWS Command Line Interface \(AWS CLI\)](#) o [AWS SDK](#), puede utilizar las siguientes operaciones de la API:

- [CreateEventSourceMapping](#)
- [ListEventSourceMappings](#)
- [GetEventSourceMapping](#)
- [UpdateEventSourceMapping](#)
- [DeleteEventSourceMapping](#)

# Controle qué eventos envía Lambda a la función

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Por ejemplo, puede agregar un filtro para que la función solo procese los mensajes de Amazon SQS que contengan ciertos parámetros de datos. El filtrado de eventos solo funciona con determinadas asignaciones de orígenes de eventos. Puede agregar filtros a las asignaciones de orígenes de eventos de los siguientes servicios de AWS:

- Amazon DynamoDB
- Amazon Kinesis Data Streams
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka autoadministrado
- Amazon Simple Queue Service (Amazon SQS)

Para obtener información específica sobre el filtrado con orígenes de eventos específicos, consulte [the section called “Uso de filtros con diferentes Servicios de AWS”](#). Lambda no es compatible con el filtrado de eventos para Amazon DocumentDB.

De manera predeterminada, puede definir hasta cinco filtros diferentes para una única asignación de orígenes de eventos. Los filtros se unen de forma lógica. Si un registro del origen de eventos cumple con uno o más de sus filtros, Lambda incluye el registro en el siguiente evento que envíe a su función. Si no se cumple con ninguno de los filtros, Lambda descarta el registro.

## Note

Si necesita definir más de cinco filtros para un origen de eventos, puede solicitar un aumento de cuota de hasta 10 filtros para cada origen de eventos. Si intenta agregar más filtros de los que permite su cuota actual, Lambda devolverá un error al intentar crear el origen de eventos.

## Temas

- [Conceptos básicos del filtrado de eventos](#)
- [Gestión de registros que no cumplen con los criterios de filtro](#)
- [Sintaxis de la regla de filtro](#)

- [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#)
- [Adjuntar criterios de filtro a una asignación de origen de eventos \(AWS CLI\)](#)
- [Adjuntar criterios de filtro a una asignación de origen de eventos \(AWS SAM\)](#)
- [Uso de filtros con diferentes Servicios de AWS](#)

## Conceptos básicos del filtrado de eventos

Un objeto de criterios de filtro (`FilterCriteria`) es una estructura que consta de una lista de filtros (`Filters`). Cada filtro es una estructura que define un patrón de filtrado de eventos (`Pattern`). Un patrón es una representación de cadenas de una regla de filtro de JSON. La estructura de un objeto `FilterCriteria` es la siguiente:

```
{
  "Filters": [
    {
      "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\":
[ rule2 ] } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "Metadata1": [ rule1 ],
  "data": {
    "Data1": [ rule2 ]
  }
}
```

El patrón de filtro puede incluir propiedades de metadatos, propiedades de datos o ambas. Los parámetros de metadatos disponibles y el formato de los parámetros de datos varían según el Servicio de AWS que actúe como origen del evento. Por ejemplo, supongamos que su asignación de orígenes de eventos recibe el siguiente registro de una cola de Amazon SQS:

```
{
  "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
  "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",
}
```

```

"body": "{\n "City": "Seattle",\n "State": "WA",\n "Temperature": "46"\n}",
"attributes": {
  "ApproximateReceiveCount": "1",
  "SentTimestamp": "1545082649183",
  "SenderId": "AIDAIENQZJOL023YVJ4V0",
  "ApproximateFirstReceiveTimestamp": "1545082649185"
},
"messageAttributes": {},
"md5ofBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
"eventSource": "aws:sqs",
"eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
"awsRegion": "us-east-2"
}

```

- Las propiedades de metadatos son los campos que contienen información sobre el evento que creó el registro. En el registro de Amazon SQS de ejemplo, las propiedades de metadatos incluyen campos como `messageID`, `eventSourceArn` y `awsRegion`.
- Las propiedades de datos son los campos del registro que contienen los datos de su flujo o cola. En el evento de Amazon SQS de ejemplo, la clave del campo de datos es `body` y las propiedades de datos son los campos `City`, `State` y `Temperature`.

Los diferentes tipos de orígenes de eventos utilizan diferentes valores de clave para sus campos de datos. Para filtrar las propiedades de datos, asegúrese de utilizar la clave correcta en el patrón del filtro. Para obtener una lista de las claves de filtrado de datos y ver ejemplos de patrones de filtro para cada uno de los Servicio de AWS compatibles, consulte [Uso de filtros con diferentes Servicios de AWS](#).

El filtrado de eventos puede gestionar el filtrado JSON multinivel. Por ejemplo, fíjese en el siguiente fragmento de un registro de un flujo de DynamoDB:

```

"dynamodb": {
  "Keys": {
    "ID": {
      "S": "ABCD"
    }
    "Number": {
      "N": "1234"
    }
  },
  ...
}

```

Supongamos que desea procesar solo los registros en los que el valor de la clave de clasificación `Number` sea 4567. En este caso, el objeto `FilterCriteria` sería similar al siguiente:

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\": [ \"4567\" ] } } } } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "dynamodb": {
    "Keys": {
      "Number": {
        "N": [ "4567" ]
      }
    }
  }
}
```

## Gestión de registros que no cumplen con los criterios de filtro

La forma en que Lambda gestiona los registros que no cumplen con los criterios del filtrado depende del origen del evento.

- En Amazon SQS, si un mensaje no cumple con los criterios de filtro, Lambda lo elimina automáticamente de la cola. No tiene que eliminar manualmente estos mensajes en Amazon SQS.
- En Kinesis y DynamoDB, después de que los criterios de filtro evalúan un registro, el iterador de flujos supera este registro. Si el registro no cumple los criterios de filtro, no tiene que eliminarlo manualmente del origen de eventos. Tras el periodo de retención, Kinesis y DynamoDB eliminan automáticamente estos registros antiguos. Si quiere que los registros se eliminen antes, consulte [Changing the Data Retention Period](#) (Cambiar el periodo de retención de datos).
- En el caso de los mensajes de Amazon MSK, Apache Kafka autoadministrado y Amazon MQ, Lambda elimina los mensajes que no coinciden con todos los campos incluidos en el filtro. Para Amazon MSK y Apache Kafka autoadministrado, Lambda compila los desplazamientos de los mensajes coincidentes y no coincidentes después de invocar la función de forma correcta. En el

caso de Amazon MQ, Lambda reconoce los mensajes coincidentes después de invocar la función de forma correcta y reconoce los mensajes no coincidentes al filtrarlos.

## Sintaxis de la regla de filtro

Para las reglas de filtro, Lambda es compatible con las reglas de Amazon EventBridge y utiliza la misma sintaxis que EventBridge. Para obtener más información, consulte [Patrones de eventos de Amazon EventBridge](#) en la Guía del usuario de Amazon EventBridge.

A continuación, se muestra un resumen de todos los operadores de comparación disponibles para el filtrado de eventos de Lambda.

Operador de comparación	Ejemplo	Sintaxis de reglas
Nulo	El valor de UserID (ID de usuario) es nulo	"UserID": [null]
Vacío	LastName (Apellido) está vacío	"LastName": [""]
Igual a	El valor de Name (Nombre) es "Alice"	"Name": ["Alice"]
Es igual a (omitir mayúsculas y minúsculas)	El valor de Name (Nombre) es "Alice"	"Name": [ { "equals-ignore-case": "alice" } ]
Y	El valor de Location (Ubicación) es "New York" y el de Day (Día) es "Monday"	"Location": ["New York"], "Day": ["Monday"]
Or (Disyunción)	El valor de PaymentType (Tipo de pago) es "Credit" (Crédito) o "Debit" (Débito)	"PaymentType": ["Credit", "Debit"]
O (varios campos)	El valor de Location (Ubicación) es "New York" o el de Day (Día) es "Monday".	"\$or": [ { "Location": [ "New York" ] }, { "Day": [ "Monday" ] } ]



Operador de comparación	Ejemplo	Sintaxis de reglas
No	El valor de Weather (Tiempo) es cualquier valor menos "Raining" (Lluvia)	"Weather": [{"anything-but": ["Raining"]}]]
Valor numérico (igual a)	El valor de Price (Precio) es 100	"Price": [{"numeric": ["=", 100]]]
Valor numérico (rango)	El valor de Price (Precio) es superior a 10 e inferior o igual a 20	"Price": [{"numeric": [ ">", 10, "<=", 20]]]
Existe	ProductName (Nombre de producto) existe	"ProductName": [{"exists": true}]
No existe	El valor de ProductName (Nombre de producto) no existe	"ProductName": [{"exists": false}]
Comienza por	El valor de Region (Región) se encuentra en US (EE. UU.)	"Region": [{"prefix": "us-"}]]
Acaba con	FileName termina con la extensión .png.	"FileName": [ { "suffix": ".png" } ]

### Note

Al igual que EventBridge, para las cadenas, Lambda usa coincidencia exacta (carácter a carácter) sin necesidad de cambio de mayúsculas y minúsculas ni cualquier otra normalización de cadenas. Para los números, Lambda también usa la representación de cadenas. Por ejemplo, 300, 300.0 y 3.0e2 no se consideran iguales.

Tenga en cuenta que el operador Exists solo funciona en los nodos hoja del origen de eventos JSON. No coincide con nodos intermedios. Por ejemplo, con el siguiente JSON, el patrón de filtro `{ "person": { "address": [ { "exists": true } ] } }` no encontraría ninguna coincidencia porque "address" es un nodo intermedio.

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
  }
}
```

## Adjuntar criterios de filtro a una asignación de origen de eventos (consola)

Siga estos pasos para crear una nueva asignación de origen de eventos con criterios de filtro mediante la consola de Lambda.

Para crear una nueva asignación de origen de eventos con criterios de filtro (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función para la que se creará una asignación de origen de eventos.
3. En Descripción general de la función, elija Agregar desencadenador.
4. En Trigger configuration (Configuración del desencadenador), elija un tipo de desencadenador compatible con el filtrado de eventos. Para obtener una lista de los servicios compatibles, consulte la lista que aparece al principio de esta página.
5. Amplíe Configuración adicional.
6. En Filter criteria (Criterios de filtro), seleccione Add (Agregar), y luego defina e ingrese los filtros. Por ejemplo, puede ingresar lo siguiente.

```
{ "Metadata" : [ 1, 2 ] }
```

De este modo, Lambda solo procesa los registros en los que el campo Metadata es igual a 1 o 2. Puede volver a seleccionar Agregar para agregar más filtros hasta la cantidad máxima permitida.

7. Cuando haya terminado de agregar filtros, elija Guardar.

Al ingresar criterios de filtro mediante la consola, solo ingresa el patrón de filtro y no necesita ingresar la clave `Pattern` ni las comillas de escape. En el paso 6 de las instrucciones anteriores, `{ "Metadata" : [ 1, 2 ] }` corresponde a los siguientes `FilterCriteria`.

```
{
  "Filters": [
    {
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
    }
  ]
}
```

Después de crear la asignación de origen de eventos en la consola, puede ver los `FilterCriteria` con formato en los detalles del desencadenador. Para obtener más ejemplos de cómo crear filtros de eventos con la consola, consulte [Uso de filtros con diferentes Servicios de AWS](#).

## Adjuntar criterios de filtro a una asignación de origen de eventos (AWS CLI)

Supongamos que quiere que una asignación de origen de eventos tenga los siguientes `FilterCriteria`:

```
{
  "Filters": [
    {
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
    }
  ]
}
```

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ] }"}]}'
```

El comando [create-event-source-mapping](#) crea una nueva asignación de orígenes de eventos de Amazon SQS para la función `my-function` con los `FilterCriteria` especificados.

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Tenga en cuenta que, para actualizar una asignación de origen de eventos, necesita su UUID. Puede obtener el UUID con una llamada a [list-event-source-mappings](#). Lambda también devuelve el UUID en la respuesta de la CLI [create-event-source-mapping](#).

Para eliminar los criterios de filtro de un origen de eventos, puede ejecutar el comando [update-event-source-mapping](#) con un objeto de `FilterCriteria` vacío.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --filter-criteria "{}"
```

Para obtener más ejemplos de cómo crear filtros de eventos con la AWS CLI, consulte [Uso de filtros con diferentes Servicios de AWS](#).

## Adjuntar criterios de filtro a una asignación de origen de eventos (AWS SAM)

Supongamos que desea configurar un origen de eventos en AWS SAM para utilizar los siguientes criterios de filtro:

```
{  
  "Filters": [  
    {  
      "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"  
    }  
  ]  
}
```

Para agregar estos criterios de filtro a su asignación de orígenes de eventos, inserte el siguiente fragmento en la plantilla YAML de su origen de eventos.

```
FilterCriteria:  
  Filters:
```

```
- Pattern: '{"Metadata": [1, 2]}'
```

Para obtener más información sobre cómo crear y configurar una plantilla AWS SAM para una asignación de orígenes de eventos, consulte la sección [EventSource](#) de la Guía para desarrolladores de AWS SAM. Para obtener más ejemplos de cómo crear filtros de eventos con AWS SAM, consulte [Uso de filtros con diferentes Servicios de AWS](#).

## Uso de filtros con diferentes Servicios de AWS

Los diferentes tipos de orígenes de eventos utilizan diferentes valores de clave para sus campos de datos. Para filtrar las propiedades de datos, asegúrese de utilizar la clave correcta en el patrón del filtro. En la siguiente tabla se muestran las claves de filtrado para cada Servicio de AWS compatible.

Servicio de AWS	Clave de filtrado
DynamoDB	dynamodb
Kinesis	data
Amazon MQ	data
Amazon MSK	value
Apache Kafka autoadministrado	value
Amazon SQS	body

En las siguientes secciones se ofrecen ejemplos de patrones de filtro para diferentes tipos de orígenes de eventos. También proporcionan definiciones de los formatos de datos entrantes compatibles y los formatos de cuerpo de los patrones de filtro para cada servicio compatible.

- [the section called “Filtrado de eventos”](#)
- [the section called “Filtrado de eventos”](#)
- [the section called “Filtrado de eventos”](#)
- [the section called “Filtrado de eventos”](#)
- [the section called “Filtrado de eventos”](#)
- [the section called “Filtrado de eventos”](#)



## Probar la función de Lambda con la consola

Puede probar la función Lambda en la consola invocando la función con un evento de prueba. Un evento de prueba es una entrada JSON a su función. Si la función no requiere una entrada, el evento puede ser un documento vacío ({}).

Cuando ejecuta una prueba en la consola, Lambda invoca su función de forma sincrónica con el evento de prueba. El tiempo de ejecución de la función convierte el JSON del evento en un objeto y lo pasa al método de controlador de su código para su procesamiento.

### Crear un evento de prueba

Antes de poder realizar la prueba en la consola, debe crear un evento de prueba privado o que se pueda compartir.

## Invocación de funciones con eventos de prueba

Para probar una función

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función que desea probar.
3. Elija la pestaña Test (Prueba).
4. En Evento de prueba, elija Crear evento nuevo o Editar evento guardado y, a continuación, elija el evento guardado que desea utilizar.
5. Si lo desea, elija una plantilla para el JSON del evento.
6. Seleccione Probar.
7. En Execution result (Resultado de ejecución), expanda Details (Detalles) para ver los resultados.

Para invocar la función sin guardar el evento de prueba, seleccione Test (Probar) antes de guardar. Así se crea un evento de prueba sin guardar que Lambda solo conservará durante la sesión.

También puede obtener acceso a sus eventos de prueba guardados y no guardados en la pestaña Code (Código). A partir de ahí, elija Test (Pruebas) y, a continuación, elija su evento de prueba.

## Creación de eventos de prueba privados

Los eventos de prueba privados solo están disponibles para el creador y no requieren permisos adicionales para utilizarlos. Puede crear hasta 10 eventos de prueba privados por función.

Si desea crear un evento de prueba

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función que desea probar.
3. Elija la pestaña Test (Prueba).
4. En Test event (Evento de prueba), haga lo siguiente:
  - a. Elija una plantilla
  - b. Introduzca un nombre para el evento de prueba.
  - c. En el cuadro de entrada de texto, introduzca el evento de prueba JSON.
  - d. En Event sharing settings (Configuración de uso compartido de eventos), elija Private (Privado).
5. Elija Guardar cambios.

También puede crear eventos de prueba nuevos en la pestaña Code (Código). A partir de ahí, elija Test (Prueba), Configure test event (Configurar evento de prueba).

## Creación de eventos de prueba compartibles

Los eventos de prueba compartibles son aquellos que puede compartir con otros usuarios en la misma cuenta de AWS. Puede editar los eventos de prueba compartibles de otros usuarios e invocar su función con ellos.

Lambda guarda los eventos de prueba que se pueden compartir como esquemas en un registro de esquemas de [Amazon EventBridge \(CloudWatch Events\) denominado](#) `lambda-testevent-schemas`. Dado que Lambda utiliza este registro para almacenar y llamar a los eventos de prueba compartibles que cree, le recomendamos que no edite este registro ni cree uno mediante el nombre `lambda-testevent-schemas`.

Para ver, compartir y editar los eventos de prueba que se pueden compartir, debe tener permisos para todas las siguientes operaciones de la API del registro de [esquemas EventBridge \(CloudWatch Events\)](#):



- [schemas.CreateRegistry](#)
- [schemas.CreateSchema](#)
- [schemas.DeleteSchema](#)
- [schemas.DeleteSchemaVersion](#)
- [schemas.DescribeRegistry](#)
- [schemas.DescribeSchema](#)
- [schemas.GetDiscoveredSchema](#)
- [schemas.ListSchemaVersions](#)
- [schemas.UpdateSchema](#)

Tenga en cuenta que guardar las ediciones realizadas a un evento de prueba que se puede compartir sobrescribe ese evento.

Si no puede crear, editar o ver eventos de prueba compartibles, compruebe que su cuenta tiene los permisos necesarios para estas operaciones. Si tienes los permisos necesarios pero sigues sin poder acceder a los eventos de prueba que se pueden compartir, comprueba si hay [políticas basadas en recursos](#) que puedan limitar el acceso al registro EventBridge (CloudWatch de eventos).

Para crear un evento de prueba compartible

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función que desea probar.
3. Elija la pestaña Test (Prueba).
4. En Test event (Evento de prueba), haga lo siguiente:
  - a. Elija una plantilla
  - b. Introduzca un nombre para el evento de prueba.
  - c. En el cuadro de entrada de texto, introduzca el evento de prueba JSON.
  - d. En Event sharing settings (Configuración de uso compartido de eventos), elija Shareable (Compartible).
5. Elija Guardar cambios.

- i** Utilice eventos de prueba que se puedan compartir con AWS Serverless Application Model. Se puede utilizar AWS SAM para invocar eventos de prueba que se pueden compartir. Consulte [sam remote test-event](#) en la [Guía de desarrolladores de AWS Serverless Application Model](#).

## Eliminación de esquemas de eventos de prueba compartibles

Al eliminar eventos de prueba compartibles, Lambda los elimina del registro de `lambda-testevent-schemas`. Si elimina el último evento de prueba compartible del registro, Lambda lo elimina.

Si elimina la función, Lambda no elimina ningún esquema de eventos de prueba compartibles asociados. Debe limpiar estos recursos manualmente desde la consola [EventBridge \(CloudWatch Events\)](#).

## Estados de función de Lambda

Lambda incluye un campo de estado en la configuración de la función para indicar cuando su función está lista. `State` proporciona información sobre el estado actual de la función e incluso si se puede invocar de forma correcta. Los estados de función no cambian el comportamiento de las invocaciones ni la forma en que la suya ejecuta el código. Los estados de función incluyen:

- **Pending**: después de que Lambda cree la función, establece el estado en pendiente. Mientras está en estado pendiente, Lambda intenta crear o configurar recursos para la función, como recursos de VPC o EFS. Lambda no invoca una función mientras está en estado pendiente. Las invocaciones u otras acciones de API que actúen en la función producirán errores.
- **Active**: la función pasa al estado activo después de que Lambda complete la configuración y el aprovisionamiento de recursos. Las funciones solo se pueden invocar correctamente mientras están activas.
- **Failed**: indica que la configuración o el aprovisionamiento de recursos ha detectado un error.
- **Inactive**: una función se vuelve inactiva cuando ha estado inactiva el tiempo suficiente para que Lambda reclame los recursos externos que se configuraron para ella. Cuando intenta invocar una función que está inactiva, la invocación produce un error y Lambda establece la función en estado pendiente hasta que se vuelvan a crear sus recursos. Si Lambda no vuelve a crear los recursos, la función vuelve al estado inactivo. Si su función está bloqueada en estado inactivo, consulte los atributos `Status Code` y `Status Code Reason` de la función para obtener más información sobre la solución de problemas. Es posible que tenga que resolver cualquier error y volver a implementar la función para restaurarla al estado activo.

Si utiliza flujos de trabajo de automatización basados en SDK o llama directamente a las API de servicio de Lambda, asegúrese de verificar que la función esté activa antes de la invocación. Para hacerlo, utilice la acción de la API de Lambda [GetFunction](#) o configure un esperador mediante el [AWS SDK for Java 2.0](#).

```
aws lambda get-function --function-name my-function --query 'Configuration.[State, LastUpdateStatus]'
```

Debería ver los siguientes datos de salida:

```
[  
  "Active",  
  "Successful"
```

```
]
```

Se produce un error en las siguientes operaciones mientras la creación de la función está pendiente:

- [Invoke](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)
- [PublishVersion](#)

## Estados de función durante la actualización

Lambda proporciona contexto adicional para las funciones que se están actualizando con el atributo `LastUpdateStatus`, que pueden tener los siguientes estados:

- `InProgress`: se está produciendo una actualización en una función existente. Mientras se está actualizando una función, las invocaciones van al código y la configuración anteriores de la función.
- `Successful`: se ha terminado la actualización. Una vez que Lambda finaliza la actualización, esta queda configurada hasta una nueva actualización.
- `Failed`: se ha producido un error al actualizar la función. Lambda anula la actualización y el código y la configuración anteriores de la función permanecen disponibles.

### Example

A continuación, se muestra el resultado del comando `get-function-configuration` en una función que se está actualizando.

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs20.x",
  "VpcConfig": {
    "SubnetIds": [
      "subnet-071f712345678e7c8",
      "subnet-07fd123456788a036",
      "subnet-0804f77612345cacf"
    ],
    "SecurityGroupIds": [
```

```
        "sg-085912345678492fb"  
    ],  
    "VpcId": "vpc-08e1234569e011e83"  
},  
"State": "Active",  
"LastUpdateStatus": "InProgress",  
...  
}
```

[FunctionConfiguration](#) tiene otros dos atributos, `LastUpdateStatusReason` y `LastUpdateStatusReasonCode`, para ayudar a solucionar problemas de actualización.

Se produce un error en las siguientes operaciones mientras se está realizando una actualización asincrónica:

- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)
- [PublishVersion](#)
- [TagResource](#)

# Comprender el comportamiento de reintento en Lambda

Al invocar una función de forma directa, se determina la estrategia para tratar los errores relacionados al código de la función. Lambda no reintenta de forma automática este tipo de errores en su nombre. Para reintentar, puede volver a invocar la función de forma manual y enviar el evento fallido a una cola para depurar o ignorar el error. El código de la función puede haberse ejecutado en su totalidad, en parte o no haberse ejecutado. Si vuelve a intentarlo, asegúrese de que el código de la función pueda gestionar el mismo evento varias veces sin generar transacciones duplicados u otros efectos colaterales no deseados.

Al invocar una función indirectamente, que debe tener en cuenta el comportamiento de reintento del invocador y cualquier servicio que la solicitud encuentra en el proceso. Esto incluye los siguientes escenarios.

- **Invocación asíncrona:** Lambda reintenta los errores de funciones dos veces. Si la función no tiene capacidad suficiente para gestionar todas las solicitudes entrantes, los eventos podrían esperar en la cola durante horas para su envío a la función. Puede configurar una cola de mensajes fallidos en la función para capturar eventos que no se hubieran procesado correctamente. Para obtener más información, consulte [Cómo Lambda administra las invocaciones asíncronas](#).
- **Mapeos de fuente de eventos:** los mapeos de fuentes de eventos que leen desde flujos reintentan el lote de elementos completo. Los errores repetidos bloquean el procesamiento del fragmento afectado hasta que se resuelve el error o los elementos caduquen. Para detectar fragmentos estancados, puede monitorizar la métrica [antigüedad del iterador](#).

Para mapeos de orígenes de eventos que leen de una cola, debe determinar el período de tiempo entre los reintentos y el destino de los eventos fallidos configurando el tiempo de espera de visibilidad y la política de redireccionamiento en la cola de origen. Para obtener más información, consulte [Cómo procesa Lambda registros de orígenes de eventos basados en secuencias y colas](#) y los temas específicos del servicio en [Invocar Lambda con eventos de otros servicios de AWS](#).

- **Servicios de AWS:** los servicios de AWS pueden invocar la función de forma [sincrónica](#) o asincrónica. Para la invocación síncrona, el servicio decide si volver a intentarlo. Por ejemplo, las operaciones por lotes de Amazon S3 reintentan la operación si la función Lambda devuelve un código de respuesta `TemporaryFailure`. Los servicios que solicitan proxy de un usuario o cliente ascendente pueden tener una estrategia de reintento o pueden transmitir la respuesta de error de vuelta al solicitante. Por ejemplo, API Gateway siempre retransmite la respuesta de error de vuelta al solicitante.

Para una invocación asíncrona, la lógica de reintento es la misma, independientemente del origen de invocación. De forma predeterminada, Lambda vuelve a intentar una invocación asíncrona fallida hasta dos veces. Para obtener más información, consulte [Cómo Lambda administra los errores y los reintentos mediante la invocación asíncrona](#).

- Otras cuentas y clientes: al conceder acceso a otras cuentas, puede utilizar [políticas basadas en recursos](#) para restringir los servicios o recursos que pueden configurar para invocar la función. Para proteger su función de sobrecargas, considere la posibilidad de colocar una capa de API delante de su función con [Amazon API Gateway](#).

Para ayudarle a abordar errores en aplicaciones de Lambda, Lambda se integra con servicios como Amazon CloudWatch y AWS X-Ray. Puede utilizar una combinación de registros, métricas, alarmas y seguimiento de para detectar e identificar rápidamente problemas en el código de la función, la API u otros recursos que admiten la aplicación. Para obtener más información, consulte [Supervisión y solución de problemas de funciones de Lambda](#).

# Utilice la detección de bucles recursivos de Lambda para evitar bucles infinitos

Cuando configura una función de Lambda para que se envíe al mismo servicio o recurso que invoca la función, es posible que se cree un bucle recursivo infinito. Por ejemplo, una función de Lambda puede escribir un mensaje en una cola de Amazon Simple Queue Service (Amazon SQS) y, a continuación, invocar la misma función. Esta invocación hace que la función escriba otro mensaje en la cola, que a su vez vuelve a invocar la función.

Los bucles recursivos involuntarios pueden hacer que se facturen cargos inesperados en su Cuenta de AWS. Los bucles también pueden hacer que Lambda [escale](#) y utilice toda la simultaneidad disponible en su cuenta. Para ayudar a reducir el impacto de los bucles involuntarios, Lambda puede detectar ciertos tipos de bucles recursivos poco después de que se produzcan. Cuando Lambda detecta un bucle recursivo, detiene la invocación de la función y se lo notifica.

Si su diseño utiliza patrones recursivos de forma intencional, puede solicitar que se desactive la detección de bucles recursivos de Lambda. Para solicitar este cambio, [póngase en contacto con AWS Support](#).

## Important

Si su diseño utiliza una función de Lambda de forma intencional para volver a escribir datos en el mismo recurso de AWS que invoca la función, tenga cuidado e implemente las medidas de protección adecuadas para evitar que se facturen cargos inesperados en su Cuenta de AWS. Para obtener más información sobre las prácticas recomendadas para usar patrones de invocación recursiva, consulte [Patrones recursivos que provocan funciones de Lambda descontroladas](#) en Serverless Land.

## Secciones

- [Descripción de la detección de bucles recursivos](#)
- [Servicios de AWS y SDK admitidos](#)
- [Notificaciones de bucle recursivo](#)
- [Cómo responder a las notificaciones de detección de bucles recursivos](#)



## Descripción de la detección de bucles recursivos

La detección de bucles recursivos de Lambda funciona mediante el seguimiento de eventos. Lambda es un servicio de computación basado en eventos que ejecuta el código de una función cuando se producen ciertos eventos. Por ejemplo, cuando se agrega un elemento a una cola de Amazon SQS o a un tema de Amazon Simple Notification Service (Amazon SNS). Lambda pasa los eventos a la función como objetos JSON, que contienen información sobre el cambio en el estado del sistema. Cuando un evento hace que la función se ejecute, esto se denomina invocación.

Para detectar bucles recursivos, Lambda usa encabezados de seguimiento de [AWS X-Ray](#). Cuando los [Servicios de AWS que admiten la detección de bucles recursivos](#) envían eventos a Lambda, esos eventos se anotan automáticamente con metadatos. Cuando la función de Lambda escribe uno de estos eventos en otro Servicio de AWS que admite una [versión compatible de un SDK de AWS](#), se actualizan los metadatos. Los metadatos actualizados incluyen un recuento del número de veces que el evento ha invocado la función.

### Note

No es necesario habilitar el seguimiento activo de X-Ray para que esta característica funcione. La detección de bucles recursivos está activada de forma predeterminada para todos los clientes de AWS. El uso de la característica es gratuito.

Una cadena de solicitudes es una secuencia de invocaciones de Lambda provocada por el mismo evento desencadenante. Por ejemplo, imagine que una cola de Amazon SQS invoca su función de Lambda. A continuación, la función de Lambda envía el evento procesado a la misma cola de Amazon SQS, que vuelve a invocar la función. En este ejemplo, cada invocación de la función se encuentra en la misma cadena de solicitudes.

Si su función se invoca más de 16 veces en la misma cadena de solicitudes, Lambda detiene automáticamente la siguiente invocación de la función en esa cadena de solicitudes y se lo notifica. Si la función está configurada con varios desencadenadores, las invocaciones de otros desencadenadores no se verán afectadas.

### Note

Cuando la configuración `maxReceiveCount` de la política de redrive de la cola de origen es superior a 16, la protección contra la recursión de Lambda no impide que Amazon SQS

vuelva a intentar el mensaje después de detectar y finalizar un bucle recursivo. Cuando Lambda detecta un bucle recursivo y descarta las invocaciones posteriores, devuelve un `RecursiveInvocationException` a la asignación de orígenes de eventos. Esto aumenta el valor `receiveCount` del mensaje. Lambda sigue reintentando el mensaje y sigue bloqueando las invocaciones de funciones hasta que Amazon SQS determine que `maxReceiveCount` se ha superado y envía el mensaje a la cola de mensajes fallidos configurada.

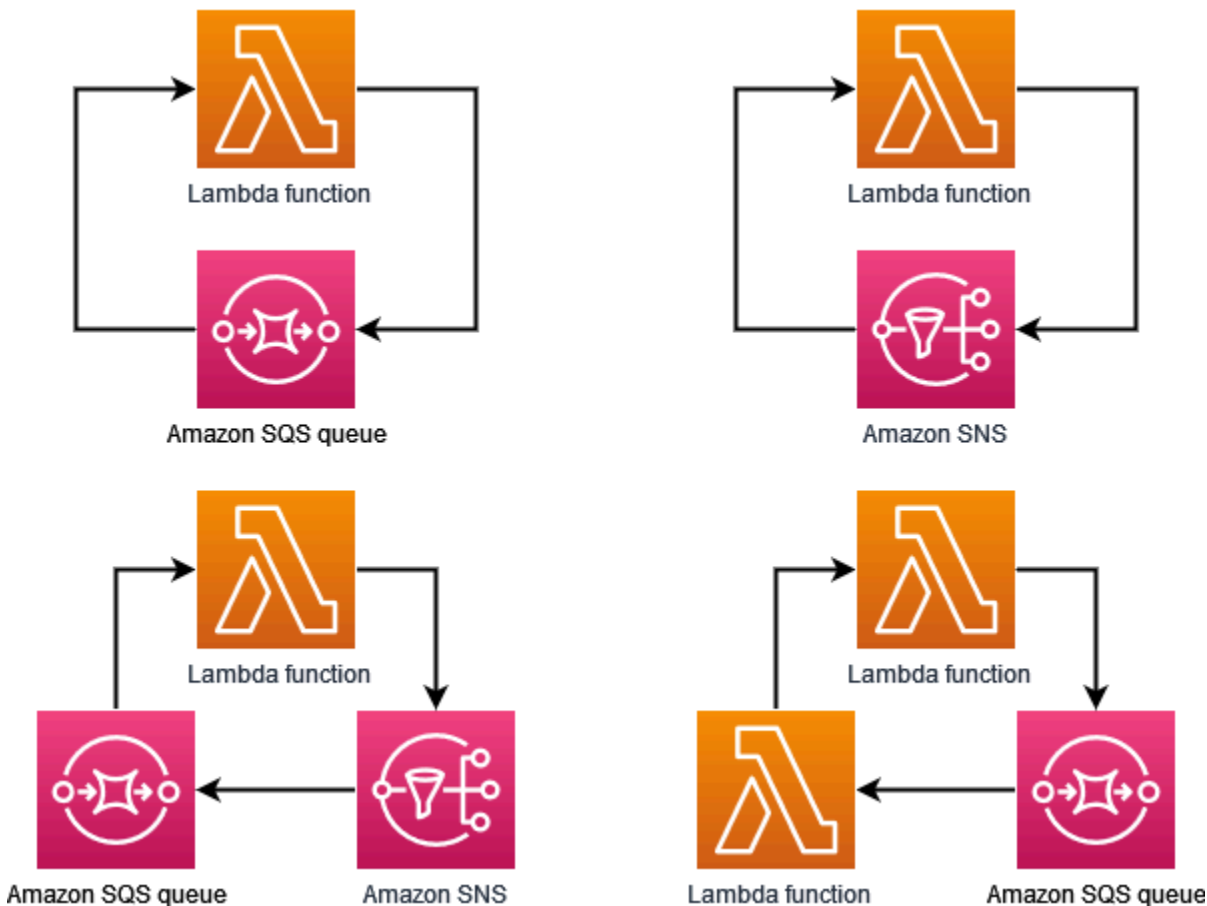
Si tiene un [destino en caso de error](#) o [una cola de mensajes fallidos](#) configurados para su función, Lambda también envía el evento desde la invocación detenida a su destino o cola de mensajes fallidos. Cuando configura un destino o una cola de mensajes fallidos para su función, asegúrese de no usar un tema de Amazon SNS o una cola de Amazon SQS que su función también use como desencadenador de eventos o asignación de orígenes de eventos. Si envía eventos al mismo recurso que invoca su función, puede crear otro bucle recursivo.

## Servicios de AWS y SDK admitidos

Lambda solo puede detectar bucles recursivos que incluyan ciertos Servicios de AWS admitidos. Para que se detecten bucles recursivos, la función también debe utilizar uno de los SDK de AWS compatibles.

### Servicios de AWS admitidos

Por el momento, Lambda detecta bucles recursivos entre sus funciones, Amazon SQS y Amazon SNS. Lambda también detecta bucles compuestos únicamente por funciones de Lambda, que pueden invocarse entre sí de forma sincrónica o asíncrona. En los siguientes diagramas, se muestran algunos ejemplos de bucles que Lambda puede detectar:



Por el momento, Lambda no puede detectar ni detener otro Servicio de AWS, como Amazon DynamoDB o Amazon Simple Storage Service (Amazon S3), cuando este forma parte del bucle.

Dado que por ahora Lambda solo detecta bucles recursivos que incluyan Amazon SQS y Amazon SNS, es posible que los bucles que incluyan otros Servicios de AWS puedan provocar un uso no deseado de las funciones de Lambda.

Para evitar que se le facturen cargos inesperados en su Cuenta de AWS, le recomendamos que configure las [alarmas de Amazon CloudWatch](#) para que le avisen sobre patrones de uso inusuales. Por ejemplo, puede configurar CloudWatch para que le notifique sobre picos en la simultaneidad o las invocaciones de las funciones de Lambda. También puede configurar una [alarma de facturación](#) para que le avise cuando el gasto en su cuenta supere el límite que especifique. También puede utilizar [AWS Cost Anomaly Detection](#) para recibir notificaciones sobre patrones de facturación inusuales.

## SDK de AWS admitidos

Para que Lambda detecte bucles recursivos, la función debe usar una de las siguientes versiones del SDK o una versión superior:

Tiempo de ejecución	Versión mínima requerida del SDK de AWS
Node.js	2.1147.0 (SDK versión 2)
	3.105.0 (SDK versión 3)
Python	1.24.46 (boto3)
	1.27.46 (botocore)
Java 8 y Java 11	1.12.200 (SDK versión 1)
	2.17.135 (SDK versión 2)
Java 17	2.20.81
Java 21	2.21.24
.NET	3.7.293.0
Ruby	3.134.0
PHP	3.232.0
Go	SDK V2 (usar la versión más reciente)

Algunos tiempos de ejecución de Lambda, como Python y Node.js, incluyen una versión del SDK de AWS. Si la versión del SDK incluida en el tiempo de ejecución de la función es inferior al mínimo requerido, puede agregar una versión compatible del SDK al [paquete de despliegue de la función](#). También puede agregar una versión compatible del SDK a la función mediante una [capa de Lambda](#). Para obtener una lista de los SDK incluidos en cada tiempo de ejecución de Lambda, consulte [Tiempos de ejecución de Lambda](#).

## Notificaciones de bucle recursivo

Cuando Lambda detiene un bucle recursivo, recibirá notificaciones por medio de [AWS Health Dashboard](#) y a través del correo electrónico. También puede usar las métricas de CloudWatch para supervisar la cantidad de invocaciones recursivas que ha detenido Lambda.

### Notificaciones de AWS Health Dashboard

Cuando Lambda detiene una invocación recursiva, AWS Health Dashboard muestra una notificación en la página Estado de su cuenta, en [Problemas abiertos y recientes](#). Tenga en cuenta que pueden pasar hasta tres horas después de que Lambda detenga una invocación recursiva antes de que se muestre esta notificación. Para obtener más información sobre cómo ver los eventos de la cuenta en AWS Health Dashboard, consulte [Introducción al Panel de AWS Health: el estado de su cuenta](#) en la Guía del usuario de AWS Health.

### Alertas por correo electrónico

Cuando Lambda detiene por primera vez una invocación recursiva de su función, le envía una alerta por correo electrónico. Lambda envía un máximo de un correo electrónico cada 24 horas para cada función de su Cuenta de AWS. Después de que Lambda envíe una notificación por correo electrónico, no recibirá más correos electrónicos para esa función hasta dentro de 24 horas, incluso si Lambda detiene nuevas invocaciones recursivas de la función. Tenga en cuenta que pueden pasar hasta tres horas después de que Lambda detenga una invocación recursiva antes de que reciba esta alerta por correo electrónico.

Lambda envía alertas por correo electrónico sobre bucles recursivos al contacto de cuenta principal de su Cuenta de AWS y al contacto de operaciones alternativas. Para obtener información sobre cómo ver o actualizar las direcciones de correo electrónico de su cuenta, consulte [Actualización de la información de contacto](#) en la Referencia general de AWS.

### Métricas de Amazon CloudWatch

La [métrica de CloudWatch](#) `RecursiveInvocationsDropped` registra el número de invocaciones de funciones que Lambda ha detenido porque la función se ha invocado más de 16 veces en una sola cadena de solicitudes. Lambda emite esta métrica tan pronto como detiene una invocación recursiva. Para ver esta métrica, siga las instrucciones de [Visualización de métricas en la consola de CloudWatch](#) y elija la métrica `RecursiveInvocationsDropped`.

## Cómo responder a las notificaciones de detección de bucles recursivos

Cuando el mismo evento desencadenante invoca la función más de 16 veces, Lambda detiene la siguiente invocación de la función para que ese evento rompa el bucle recursivo. Para evitar que vuelva a aparecer un bucle recursivo que Lambda ha roto, haga lo siguiente:

- Reduzca la [simultaneidad](#) disponible de la función a cero, lo que limita todas las invocaciones futuras.
- Elimine o deshabilite el desencadenador o la asignación de orígenes de eventos que invoca su función.
- Identifique y corrija los defectos del código que devuelven los eventos al recurso de AWS que invoca la función. Un origen común de defectos se produce cuando se utilizan variables para definir el origen y el destino de los eventos de una función. Compruebe que no está usando el mismo valor para ambas variables.

Además, si el origen de eventos de la función de Lambda es una cola de Amazon SQS, considere la posibilidad de [configurar una cola de mensajes fallidos](#) en la cola de origen.

### Note

Asegúrese de configurar la cola de mensajes fallidos en la cola de origen y no en la función de Lambda. La cola de mensajes fallidos que configure en una función se utiliza para la [cola de invocación asíncrona](#) de la función, no para las colas de origen de eventos.

Si el origen del evento es un tema de Amazon SNS, considere agregar un [destino en caso de error](#) para su función.

Para reducir a cero la simultaneidad disponible de la función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de su función.
3. Elija Limitar.
4. En el cuadro de diálogo Limitar la función, seleccione Confirmar.

Para eliminar un desencadenador o una asignación de orígenes de eventos de la función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de su función.
3. Elija la pestaña Configuración y, a continuación, seleccione Desencadenadores.
4. En Desencadenadores, seleccione el desencadenador o la asignación de orígenes de eventos que desee eliminar y, a continuación, seleccione Eliminar.
5. En el cuadro de diálogo Eliminar desencadenadores, seleccione Eliminar.

Para deshabilitar una asignación de orígenes de eventos de la función (AWS CLI)

1. Para encontrar el UUID de la asignación de orígenes de eventos que desea deshabilitar, ejecute el comando de AWS Command Line Interface (AWS CLI) [list-event-source-mappings](#).

```
aws lambda list-event-source-mappings
```

2. Para deshabilitar la asignación de orígenes de eventos, ejecute el siguiente comando de AWS CLI [update-event-source-mapping](#).

```
aws lambda update-event-source-mapping --function-name MyFunction \  
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

## Creación y administración de URL de funciones de Lambda

Una URL de función es un punto de conexión HTTP(S) dedicado para la función de Lambda. Puede crear y configurar una URL de función a través de la consola de Lambda o la API de Lambda. Al crear una URL de función, Lambda genera automáticamente un punto de conexión de URL único para usted. Una vez que crea una URL de función, el punto de conexión de la URL nunca cambia. Los puntos de conexión de la URL de función tienen el siguiente formato:

```
https://<url-id>.lambda-url.<region>.on.aws
```

### Note

Las URL de función no se admiten en las siguientes regiones: Asia-Pacífico (Hyderabad) (ap-south-2), Asia-Pacífico (Melbourne) (ap-southeast-4), Oeste de Canadá (Calgary) (ca-west-1) Europa (España) (eu-south-2), Europa (Zúrich) (eu-central-2), Israel (Tel Aviv) (il-central-1) y Medio Oriente (Emiratos Árabes Unidos) (me-central-1).

Las URL de funciones están habilitadas para doble pila y son compatibles con IPv4 e IPv6. Después de configurar una URL de función para su función, puede invocar la función a través de su punto de conexión HTTP(S) mediante un navegador web, curl, Postman o cualquier cliente HTTP.

### Note

Puede acceder a la URL de la función solo a través de la Internet pública. Si bien las funciones de Lambda son compatibles con AWS PrivateLink, las URL de las funciones no lo son.

Las URL de funciones de Lambda utilizan [políticas basadas en recursos](#) para la seguridad y el control de acceso. Las URL de funciones también admiten opciones de configuración de uso compartido de recursos entre orígenes (CORS).

Puede aplicar las URL de las funciones a cualquier alias de la función o a la versión \$LATEST de la función no publicada. No se puede agregar una URL de función a ninguna otra versión de la función.

La siguiente sección muestra cómo crear y administrar una URL de función mediante la consola de Lambda, la AWS CLI y la plantilla de AWS CloudFormation.



## Temas

- [Creación de una URL de función \(consola\)](#)
- [Creación de una URL de función \(AWS CLI\)](#)
- [Adición de una URL de función a una plantilla de CloudFormation](#)
- [Uso compartido de recursos entre orígenes \(CORS\)](#)
- [URL de funciones de limitación](#)
- [Desactivación de URL de funciones](#)
- [Eliminación de URL de función](#)
- [Control de acceso a las URL de las funciones de Lambda](#)
- [Invocación de URL de funciones de Lambda](#)
- [Supervisión de las URL de funciones de Lambda](#)
- [Tutorial: Creación de una función de Lambda con una URL de función](#)

## Creación de una URL de función (consola)

Siga estos pasos para crear una URL de función mediante la consola.

Para crear una URL de función para una función existente (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función para la que desea crear la URL de función.
3. Elija la pestaña Configuration (Configuración) y, a continuación, elija Function URL (URL de función).
4. Elija Create function URL (Crear URL de función).
5. Para el Auth type (Tipo de autenticación), elija AWS\_IAM o NONE (NINGUNO). Para obtener más información sobre la autenticación de URL de función, consulte [Control de acceso](#).
6. (Opcional) Seleccione Configure cross-origin resource sharing (CORS) (Configuración de uso compartido de recursos entre orígenes [CORS]) y, a continuación, configure los ajustes de CORS para la URL de función. Para obtener más información acerca de CORS, consulte [Uso compartido de recursos entre orígenes \(CORS\)](#).
7. Seleccione Guardar.

Esto crea una URL de función para la versión \$LATEST no publicada de la función. La URL de función aparece en la sección Function overview (Información general de la función) de la consola.

Para crear una URL de función para un alias existente (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función con el alias para el que desea crear la URL de función.
3. Elija la pestaña Aliases (Alias) y, a continuación, elija el nombre del alias para el que desea crear la URL de función.
4. Elija la pestaña Configuration (Configuración) y, a continuación, elija URL de función (URL de función).
5. Elija Create function URL (Crear URL de función).
6. Para el Auth type (Tipo de autenticación), elija AWS\_IAM o NONE (NINGUNO). Para obtener más información sobre la autenticación de URL de función, consulte [Control de acceso](#).
7. (Opcional) Seleccione Configure cross-origin resource sharing (CORS) (Configuración de uso compartido de recursos entre orígenes [CORS]) y, a continuación, configure los ajustes de CORS para la URL de función. Para obtener más información acerca de CORS, consulte [Uso compartido de recursos entre orígenes \(CORS\)](#).
8. Seleccione Guardar.

Esto crea una URL de función para el alias de la función. La URL de función aparece en la sección Información general de la función para su alias.

Para crear una nueva función con una URL de función (consola)

Para crear una nueva función con una URL de función (consola)

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija Create function (Crear función).
3. Bajo Basic information (Información básica), haga lo siguiente:
  - a. En Function name (Nombre de la función), ingrese un nombre para la función, como **my-function**.
  - b. En Runtime (Tiempo de ejecución), elija el tiempo de ejecución del lenguaje que prefiera, como Node.js 18.x.
  - c. En Architecture (Arquitectura), elija x86\_64 o arm64.

- d. Expanda Permissions (Permisos) y, a continuación, elija si desea crear un nuevo rol de ejecución o utilizar uno existente.
4. Expanda Advanced settings (Configuración avanzada) y, a continuación, seleccione Function URL (URL de función).
5. Para el Auth type (Tipo de autenticación), elija AWS\_IAM o NONE (NINGUNO). Para obtener más información sobre la autenticación de URL de función, consulte [Control de acceso](#).
6. (Opcional) Seleccione Configure cross-origin resource sharing (CORS) (Configuración de uso compartido de recursos entre orígenes [CORS]). Al seleccionar esta opción durante la creación de la función, la URL de función permite solicitudes de todos los orígenes de forma predeterminada. Puede editar la configuración de CORS de la URL de función después de crear la función. Para obtener más información acerca de CORS, consulte [Uso compartido de recursos entre orígenes \(CORS\)](#).
7. Elija Crear función.

Esto crea una nueva función con una URL de función para la versión \$LATEST no publicada de la función. La URL de función aparece en la sección Function overview (Información general de la función) de la consola.

## Creación de una URL de función (AWS CLI)

Para crear una URL de función para una función de Lambda existente mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando:

```
aws lambda create-function-url-config \  
  --function-name my-function \  
  --qualifier prod \ // optional  
  --auth-type AWS_IAM  
  --cors-config {AllowOrigins="https://example.com"} // optional
```

Esto añade una URL de función al calificador **prod** para la función **my-function**. Para obtener más información acerca de estos parámetros de configuración, consulte [CreateFunctionUrlConfig](#) en la referencia de la API.

### Note

Para crear una URL de función mediante la AWS CLI, la función ya debe existir.

## Adición de una URL de función a una plantilla de CloudFormation

Para agregar un recurso de `AWS::Lambda::Url` a su plantilla de AWS CloudFormation, utilice la siguiente sintaxis:

### JSON

```
{
  "Type" : "AWS::Lambda::Url",
  "Properties" : {
    "AuthType" : String,
    "Cors" : Cors,
    "Qualifier" : String,
    "TargetFunctionArn" : String
  }
}
```

### YAML

```
Type: AWS::Lambda::Url
Properties:
  AuthType: String
  Cors:
    Cors
  Qualifier: String
  TargetFunctionArn: String
```

### Parámetros

- (Obligatorio) `AuthType`: define el tipo de autenticación de la URL de función. Los valores posibles son `AWS_IAM` o `NONE`. Para restringir el acceso solo a los usuarios autenticados, establézcalo en `AWS_IAM`. Para omitir la autenticación de IAM y permitir que cualquier usuario realice solicitudes a su función, establézcalo en `NONE`.
- (Opcional) `Cors`: define la [configuración de CORS](#) para la URL de función. Para agregar `Cors` al recurso de `AWS::Lambda::Url` en CloudFormation, utilice la siguiente sintaxis.

#### Example `AWS::Lambda::Url.Cors` (JSON)

```
{
```

```
"AllowCredentials" : Boolean,  
"AllowHeaders" : [ String, ... ],  
"AllowMethods" : [ String, ... ],  
"AllowOrigins" : [ String, ... ],  
"ExposeHeaders" : [ String, ... ],  
"MaxAge" : Integer  
}
```

### Example AWS::Lambda::Url.Cors (YAML)

```
AllowCredentials: Boolean  
AllowHeaders:  
  - String  
AllowMethods:  
  - String  
AllowOrigins:  
  - String  
ExposeHeaders:  
  - String  
MaxAge: Integer
```

- (Opcional) **Qualifier**: el nombre del alias.
- (Obligatorio) **TargetFunctionArn**: el nombre o nombre de recurso de Amazon (ARN) de la función de Lambda. Los formatos de nombre válidos incluyen los siguientes:
  - Nombre de la función: `my-function`.
  - ARN de la función: `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
  - ARN parcial: `123456789012:function:my-function`.

## Uso compartido de recursos entre orígenes (CORS)

Para definir cómo los distintos orígenes pueden acceder a la URL de función, utilice el [uso compartido de recursos entre orígenes \(CORS\)](#). Recomendamos configurar CORS si tiene la intención de llamar a la URL de función desde otro dominio. Lambda es compatible con los siguientes encabezados CORS para las URL de funciones.

Encabezado CORS	Propiedad de configuración de CORS	Valores de ejemplo
<a href="#">Access-Control-Allow-Origin</a>	AllowOrigins	* (permitir todos los orígenes)  https://www.example.com  http://localhost:60905
<a href="#">Access-Control-Allow-Methods</a>	AllowMethods	GET, POST, DELETE, *
<a href="#">Access-Control-Allow-Headers</a>	AllowHeaders	Date, Keep-Alive, X-Custom-Header
<a href="#">Access-Control-Expose-Headers</a>	ExposeHeaders	Date, Keep-Alive, X-Custom-Header
<a href="#">Access-Control-Allow-Credentials</a>	AllowCredentials	TRUE
<a href="#">Access-Control-Max-Age</a>	MaxAge	5 (predeterminado), 300

Al configurar CORS para una URL de función mediante la consola de Lambda o la AWS CLI, Lambda agrega automáticamente los encabezados CORS a todas las respuestas a través de la URL de función. También puede agregar manualmente encabezados CORS a la respuesta de la función. Si hay encabezados contradictorios, el comportamiento esperado depende del tipo de solicitud:

- Para las solicitudes de verificación previa, como las solicitudes OPTIONS, prevalecen los encabezados CORS configurados en la URL de función. Lambda devuelve solo estos encabezados CORS en la respuesta.
- Para las solicitudes que no son de verificación previa, como las solicitudes GET o POST, Lambda devuelve tanto los encabezados CORS configurados en la URL de función como los encabezados CORS devueltos por la función. Esto puede provocar encabezados CORS duplicados en la respuesta. Es posible que vea un error como el siguiente: The 'Access-Control-Allow-Origin' header contains multiple values '\*', '\*', but only one is allowed.

En general, se recomienda configurar todos los ajustes de CORS en la URL de función, en lugar de enviar los encabezados CORS manualmente en la respuesta de la función.

## URL de funciones de limitación

La limitación limita la velocidad a la que la función procesa las solicitudes. Esto resulta útil en muchas situaciones, como evitar que la función sobrecargue los recursos descendentes o gestionar un aumento repentino de las solicitudes.

Puede limitar la tasa de solicitudes que procesa la función de Lambda a través de una URL de función configurando la simultaneidad reservada. La simultaneidad reservada limita el número de invocaciones simultáneas máximas para la función. La tasa máxima de solicitudes por segundo (RPS) de la función equivale a 10 veces la simultaneidad reservada configurada. Por ejemplo, si configura la función con una simultaneidad reservada de 100, el RPS máximo es de 1000.

Cada vez que la simultaneidad de la función supera la simultaneidad reservada, la URL de función devuelve un código de estado HTTP 429. Si la función recibe una solicitud que supera 10 veces el máximo de RPS en función de la simultaneidad reservada configurada, también recibirá un error HTTP 429. Para obtener más información acerca de la simultaneidad reservada, consulte [Configurar la simultaneidad reservada para una función](#).

## Desactivación de URL de funciones

En caso de emergencia, es posible que quiera rechazar todo el tráfico a la URL de función. Para desactivar la URL de función, establezca la simultaneidad reservada en cero. Esto limita todas las solicitudes a la URL de función, lo que da como resultado respuestas de estado HTTP 429. Para reactivar la URL de función, elimine la configuración de simultaneidad reservada o establezca la configuración en un importe superior a cero.

## Eliminación de URL de función

Al eliminar una URL de función, no se la puede recuperar. La creación de una nueva URL de función dará como resultado una dirección URL diferente.

### Note

Si elimina una URL de función con el tipo de autenticación NONE, Lambda no elimina de forma automática la política basada en recursos asociada. Si desea eliminar esta política, debe hacerlo de forma manual.

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función.
3. Elija la pestaña Configuration (Configuración) y, a continuación, elija Function URL (URL de función).
4. Elija Eliminar.
5. Escriba la palabra delete (eliminar) en el campo para confirmar la eliminación.
6. Elija Eliminar.

#### Note

Al eliminar una función que tiene una URL de función, Lambda la elimina de forma asíncrona. Si crea inmediatamente una función nueva con el mismo nombre en la misma cuenta, es posible que la URL de la función original se asigne a la nueva función en lugar de eliminarse.

## Control de acceso a las URL de las funciones de Lambda

Puede controlar el acceso a las URL de funciones de Lambda mediante el parámetro AuthType combinado con [políticas basadas en recursos](#) adjuntas a la función específica. La configuración de estos dos componentes determina quién puede invocar o realizar otras acciones administrativas en la URL de función.

El parámetro AuthType determina cómo Lambda autentica o autoriza las solicitudes a la URL de función. Al configurar la URL de función, debe especificar una de las siguientes opciones de AuthType:

- **AWS\_IAM:** Lambda utiliza AWS Identity and Access Management (IAM) para autenticar y autorizar solicitudes basadas en la política de identidad de la entidad principal de IAM y la política basada en recursos de la función. Elija esta opción si desea que solo los usuarios y roles de autenticados invoquen su función a través de la URL de función.
- **NONE:** Lambda no realiza ninguna autenticación antes de invocar la función. Sin embargo, la política basada en recursos de la función siempre está vigente y debe conceder acceso público para que la URL de función pueda recibir solicitudes. Elija esta opción para permitir el acceso público y no autenticado a la URL de función.



Además de AuthType, puede utilizar políticas basadas en recursos para conceder permisos a otras Cuentas de AWS para invocar su función. Para obtener más información, consulte [Trabajar con políticas de IAM basadas en recursos en Lambda](#).

Para obtener más información sobre la seguridad, puede utilizar AWS Identity and Access Management Access Analyzer para obtener un análisis exhaustivo del acceso externo a la URL de función. IAM Access Analyzer también supervisa los permisos nuevos o actualizados en las funciones de Lambda para ayudarle a identificar los permisos que otorgan acceso público y entre cuentas. IAM Access Analyzer es de uso gratuito para cualquier cliente de AWS. Para comenzar a utilizar IAM Access Analyzer, consulte [Uso de AWS IAM Access Analyzer](#).

Esta página contiene ejemplos de políticas basadas en recursos para ambos tipos de autenticación y también cómo crear estas políticas mediante la operación de la API [AddPermission](#) o la consola de Lambda. Para obtener información sobre cómo invocar la URL de función después de configurar los permisos, consulte [Invocación de URL de funciones de Lambda](#).

## Temas

- [Uso del tipo de autenticación AWS\\_IAM](#)
- [Uso del tipo de autenticación NONE](#)
- [Gobierno y control de acceso](#)

## Uso del tipo de autenticación **AWS\_IAM**

Si elige el tipo de autenticación AWS\_IAM, los usuarios que necesiten invocar la URL de función de Lambda deben tener el permiso `lambda:InvokeFunctionUrl`. Según quién realice la solicitud de invocación, es posible que deba conceder este permiso mediante una política basada en recursos.

Si la entidad principal que realiza la solicitud está en la misma Cuenta de AWS que la URL de función, entonces la entidad principal debe tener o bien permisos `lambda:InvokeFunctionUrl` en su [política basada en identidad](#), o bien permisos concedidos en la política basada en recursos de la función. En otras palabras, una política basada en recursos es opcional si el usuario ya tiene permisos `lambda:InvokeFunctionUrl` en su política basada en identidad. La evaluación de políticas sigue las reglas descritas en [Cómo determinar si una solicitud se permite o se deniega dentro de una cuenta](#).

Si la entidad principal que realiza la solicitud se encuentra en una cuenta diferente, la entidad principal debe tener tanto una política basada en identidad que le otorgue permisos

`lambda:InvokeFunctionUrl` como permisos concedidos en una política basada en recursos sobre la función que intenta invocar. En estos casos entre cuentas, la evaluación de políticas sigue las reglas descritas en [Determinación de si se permite una solicitud entre cuentas](#).

Para ver un ejemplo de interacción entre cuentas, la siguiente política basada en recursos permite al rol `example` en la Cuenta de AWS `444455556666` invocar la URL de función asociada a la función `my-function`:

Example política de invocación entre cuentas de la URL de función

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
      }
    }
  ]
}
```

Puede crear esta instrucción de política a través de la consola siguiendo estos pasos:

Para conceder permisos de invocación de URL a otra cuenta (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de la función para la que desea conceder permisos de invocación de la URL.
3. Elija la pestaña Configuration (Configuración) y, a continuación, elija Permissions (Permisos).
4. En Resource-based policy (Política basada en recursos), elija Add permissions (Agregar permisos).
5. Elija Function URL (URL de función).
6. En Auth type (Tipo de autenticación), elija AWS\_IAM.

7. (Opcional) En Statement ID (ID de instrucción), ingrese un ID de instrucción para su instrucción de política.
8. En Entidad principal, ingrese el ID de cuenta o el nombre de recurso de Amazon (ARN) del usuario o rol al que desea conceder permisos. Por ejemplo: **444455556666**.
9. Seleccione Guardar.

Como alternativa, puede crear esta instrucción de política mediante el comando [add-permission](#) de AWS Command Line Interface (AWS CLI) siguiente:

```
aws lambda add-permission --function-name my-function \  
  --statement-id example0-cross-account-statement \  
  --action lambda:InvokeFunctionUrl \  
  --principal 444455556666 \  
  --function-url-auth-type AWS_IAM
```

En el ejemplo anterior, el valor de clave de la condición `lambda:FunctionUrlAuthType` es `AWS_IAM`. Esta política solo permite el acceso cuando el tipo de autenticación de la URL de función también es `AWS_IAM`.

## Uso del tipo de autenticación **NONE**

### Important

Cuando el tipo de autenticación de la URL de función es `NONE` y tiene una política basada en recursos que concede acceso público, cualquier usuario no autenticado con la URL de función puede invocar la función.

En algunos casos, es posible que desee que la URL de función sea pública. Por ejemplo, es posible que desee atender solicitudes realizadas directamente desde un navegador web. Para permitir el acceso público a la URL de función, elija el tipo de autenticación `NONE`.

Si elige el tipo de autenticación `NONE`, Lambda no utilizará IAM para autenticar las solicitudes a la URL de función. Sin embargo, los usuarios deben tener permisos `lambda:InvokeFunctionUrl` para invocar correctamente la URL de función. Puede conceder permisos `lambda:InvokeFunctionUrl` utilizando la siguiente política basada en recursos:

## Example política de invocación de URL de función para todas las entidades principales no autenticadas

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "NONE"
        }
      }
    }
  ]
}
```

### Note

Al crear una URL de función con tipo de autenticación NONE a través de la consola o AWS Serverless Application Model (AWS SAM), Lambda crea automáticamente la instrucción de política basada en recursos anterior por usted. Si la política ya existe o el usuario o el rol que crea la aplicación no tiene los permisos adecuados, Lambda no creará la política por usted. Si utiliza la AWS CLI, AWS CloudFormation o la API de Lambda directamente, debe agregar los permisos `lambda:InvokeFunctionUrl` usted. Esto hace que su función sea pública. Además, si elimina la URL de función con el tipo de autenticación NONE, Lambda no elimina de forma automática la política basada en recursos asociada. Si desea eliminar esta política, debe hacerlo de forma manual.

En esta instrucción, el valor de clave de la condición `lambda:FunctionUrlAuthType` es NONE. Esta instrucción de política solo permite el acceso cuando el tipo de autenticación de la URL de función también es NONE.

Si la política basada en recursos de una función no concede permisos `lambda:invokeFunctionUrl`, los usuarios obtendrán un código de error 403 Forbidden

(Prohibido) cuando intenten invocar la URL de función, incluso si la URL de función utiliza el tipo de autenticación NONE.

## Gobierno y control de acceso

Además de los permisos de invocación de la URL de función, también puede controlar el acceso a las acciones utilizadas para configurar las URL de funciones. Lambda es compatible con las siguientes acciones de política de IAM para las URL de funciones:

- `lambda:InvokeFunctionUrl`: invocar una función de Lambda mediante la URL de función.
- `lambda:CreateFunctionUrlConfig`: crear una URL de función y configurar su `AuthType`.
- `lambda:UpdateFunctionUrlConfig`: actualizar una configuración de URL de función y su `AuthType`.
- `lambda:GetFunctionUrlConfig`: ver los detalles de una URL de función.
- `lambda:ListFunctionUrlConfigs`: enumerar las configuraciones de la URL de función.
- `lambda>DeleteFunctionUrlConfig`: eliminar una URL de función.

### Note

La consola de Lambda admite agregar permisos solo para `lambda:InvokeFunctionUrl`. Para todas las demás acciones, debe agregar permisos mediante la API de Lambda o AWS CLI.

Para permitir o denegar el acceso de otras entidades de AWS a la URL de función, incluya estas acciones en las políticas de IAM. Por ejemplo, la siguiente política concede permisos al rol `example` en la Cuenta de AWS 444455556666 para actualizar la URL de función para la función **my-function** en la cuenta 123456789012.

### Example política de URL de función entre cuentas

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```

        "AWS": "arn:aws:iam::444455556666:role/example"
    },
    "Action": "lambda:UpdateFunctionUrlConfig",
    "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
}
]
}

```

## Claves de condición

Para obtener un control de acceso detallado sobre las URL de funciones, utilice una clave de condición. Lambda admite una clave de condición adicional para las URL de funciones: `FunctionUrlAuthType`. La clave `FunctionUrlAuthType` define un valor de enumeración que describe el tipo de autenticación que utiliza la URL de función. El valor puede ser `AWS_IAM` o `NONE`.

Puede utilizar esta clave de condición en las políticas asociadas a la función. Por ejemplo, es posible que desee restringir quién puede realizar cambios de configuración en las URL de funciones. Para denegar todas las solicitudes `UpdateFunctionUrlConfig` a cualquier función con tipo de autenticación de URL `NONE`, puede definir la siguiente política:

## Example política de URL de función con denegación explícita

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "lambda:UpdateFunctionUrlConfig"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "NONE"
        }
      }
    }
  ]
}

```

Para conceder permisos al rol `example` en la Cuenta de AWS `444455556666` para hacer solicitudes `CreateFunctionUrlConfig` y `UpdateFunctionUrlConfig` en funciones con tipo de autenticación de URL `AWS_IAM`, puede definir la siguiente política:

#### Example política de URL de función con permiso explícito

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": [
        "lambda:CreateFunctionUrlConfig",
        "lambda:UpdateFunctionUrlConfig"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
      }
    }
  ]
}
```

También puede utilizar esta clave de condición en una [política de control de servicios](#) (SCP).

Utilice las SCP para administrar los permisos en toda una organización en AWS Organizations. Por ejemplo, para denegar a los usuarios la creación o actualización de URL de funciones que utilicen un tipo de autenticación distinto a `AWS_IAM`, utilice la siguiente política de control de servicios:

#### Example SCP de URL de función con denegación explícita

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "lambda:CreateFunctionUrlConfig",

```

```
        "lambda:UpdateFunctionUrlConfig"
    ],
    "Resource": "arn:aws:lambda:*:123456789012:function:*",
    "Condition": {
        "StringNotEquals": {
            "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
    }
}
]
```



## Invocación de URL de funciones de Lambda

Una URL de función es un punto de conexión HTTP(S) dedicado para la función de Lambda. Puede crear y configurar una URL de función a través de la consola de Lambda o la API de Lambda. Al crear una URL de función, Lambda genera automáticamente un punto de conexión de URL único para usted. Una vez que crea una URL de función, el punto de conexión de la URL nunca cambia. Los puntos de conexión de la URL de función tienen el siguiente formato:

```
https://<url-id>.lambda-url.<region>.on.aws
```

### Note

Las URL de función no se admiten en las siguientes regiones: Asia-Pacífico (Hyderabad) (ap-south-2), Asia-Pacífico (Melbourne) (ap-southeast-4), Oeste de Canadá (Calgary) (ca-west-1) Europa (España) (eu-south-2), Europa (Zúrich) (eu-central-2), Israel (Tel Aviv) (il-central-1) y Medio Oriente (Emiratos Árabes Unidos) (me-central-1).

Las URL de funciones están habilitadas para doble pila y son compatibles con IPv4 e IPv6. Después de configurar la URL de función, puede invocar la función a través de su punto de conexión HTTP(S) mediante un navegador web, curl, Postman o cualquier cliente HTTP. Para invocar una URL de función, debe tener permisos `lambda:InvokeFunctionUrl`. Para obtener más información, consulte [Control de acceso](#).

### Temas

- [Conceptos básicos de invocación de URL de funciones](#)
- [Cargas de solicitud y respuesta](#)

## Conceptos básicos de invocación de URL de funciones

Si la URL de función utiliza el tipo de autenticación `AWS_IAM`, debe firmar cada solicitud HTTP utilizando [AWS Signature Version 4 \(SigV4\)](#). Herramientas tales como [awsurl](#), [Postman](#) y [AWS SigV4 Proxy](#) ofrecen formas integradas de firmar sus solicitudes con SigV4.

Si no utiliza una herramienta para firmar solicitudes HTTP en la URL de función, debe firmar manualmente cada solicitud mediante SigV4. Cuando la URL de función recibe una solicitud, Lambda también calcula la firma SigV4. Lambda procesa la solicitud solo si las firmas coinciden. Para obtener

instrucciones sobre cómo firmar de manera manual las solicitudes con SigV4, consulte [Firmar solicitudes de AWS con Signature Version 4](#) en la Guía de Referencia general de Amazon Web Services.

Si la URL de función utiliza el tipo de autenticación NONE, no es necesario que firme las solicitudes con SigV4. Puede invocar la función mediante un navegador web, curl, Postman o cualquier cliente HTTP.

Para probar solicitudes GET simples a su función, utilice un navegador web. Por ejemplo, si la URL de función es `https://abcdefg.lambda-url.us-east-1.on.aws`, y toma un parámetro de cadena message, la URL de la solicitud podría tener un aspecto similar al siguiente:

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Para probar otras solicitudes HTTP, como una solicitud POST, puede utilizar una herramienta como curl. Por ejemplo, si desea incluir algunos datos JSON en una solicitud POST a la URL de función, puede utilizar el siguiente comando curl:

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \  
-H 'content-type: application/json' \  
-d '{ "example": "test" }'
```

## Cargas de solicitud y respuesta

Cuando un cliente llama a la URL de función, Lambda asigna la solicitud a un objeto de evento antes de pasarla a la función. A continuación, la respuesta de la función se asigna a una respuesta HTTP que Lambda envía de vuelta al cliente a través de la URL de función.

Los formatos de eventos de solicitud y respuesta siguen el mismo esquema que la [versión de formato de carga 2.0 de Amazon API Gateway](#).

### Formato de carga de solicitud

La carga de una solicitud tiene la siguiente estructura:

```
{  
  "version": "2.0",  
  "routeKey": "$default",  
  "rawPath": "/my/path",  
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",  
  "cookies": [  
    {  
      "name": "cookie1",  
      "value": "value1",  
      "path": "/my/path",  
      "domain": "example.com",  
      "expires": "2020-01-01T00:00:00.000Z",  
      "secure": true,  
      "httpOnly": true,  
      "sameSite": "strict"  
    },  
    {  
      "name": "cookie2",  
      "value": "value2",  
      "path": "/my/path",  
      "domain": "example.com",  
      "expires": "2020-01-01T00:00:00.000Z",  
      "secure": true,  
      "httpOnly": true,  
      "sameSite": "strict"  
    }  
  ]  
}
```

```
"cookie1",
"cookie2"
],
"headers": {
  "header1": "value1",
  "header2": "value1,value2"
},
"queryStringParameters": {
  "parameter1": "value1,value2",
  "parameter2": "value"
},
"requestContext": {
  "accountId": "123456789012",
  "apiId": "<urlid>",
  "authentication": null,
  "authorizer": {
    "iam": {
      "accessKey": "AKIA...",
      "accountId": "111122223333",
      "callerId": "AIDA...",
      "cognitoIdentity": null,
      "principalOrgId": null,
      "userArn": "arn:aws:iam::111122223333:user/example-user",
      "userId": "AIDA..."
    }
  },
  "domainName": "<url-id>.lambda-url.us-west-2.on.aws",
  "domainPrefix": "<url-id>",
  "http": {
    "method": "POST",
    "path": "/my/path",
    "protocol": "HTTP/1.1",
    "sourceIp": "123.123.123.123",
    "userAgent": "agent"
  },
  "requestId": "id",
  "routeKey": "$default",
  "stage": "$default",
  "time": "12/Mar/2020:19:03:58 +0000",
  "timeEpoch": 1583348638390
},
"body": "Hello from client!",
"pathParameters": null,
"isBase64Encoded": false,
```

```
"stageVariables": null
}
```

Parámetro	Descripción	Ejemplo
<code>version</code>	La versión de formato de carga de este evento. Actualmente, las URL de funciones de Lambda son compatibles con la <a href="#">versión de formato de carga 2.0</a> .	2.0
<code>routeKey</code>	Las URL de funciones no utilizan este parámetro. Lambda establece <code>\$default</code> como marcador de posición.	<code>\$default</code>
<code>rawPath</code>	Ruta de acceso de la solicitud. Por ejemplo, si la URL de solicitud es <code>https://{url-id}.lambda-url.{region}.on.aws/example/test/demo</code> , el valor de la ruta sin procesar es <code>/example/test/demo</code> .	<code>/example/test/demo</code>
<code>rawQueryString</code>	La cadena sin procesar que contiene los parámetros de cadena de consulta de la solicitud. Los caracteres admitidos incluyen a-z, A-Z, 0-9, ., _, -, %, &, = y +.	<code>"?parameter1=value1&amp;parameter2=value2"</code>
<code>cookies</code>	Una matriz que contiene todas las cookies enviadas como parte de la solicitud.	<code>["Cookie_1=Value_1", "Cookie_2=Value_2"]</code>

Parámetro	Descripción	Ejemplo
<code>headers</code>	La lista de encabezados de solicitud, presentada como pares clave-valor.	<pre>{"header1": "value1", "header2": "value2"}</pre>
<code>queryStringParameters</code>	Los parámetros de consulta de la solicitud. Por ejemplo, si la URL de solicitud es <code>https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane</code> , el valor <code>queryStringParameters</code> es un objeto JSON con una clave de <code>name</code> y un valor de <code>Jane</code> .	<pre>{"name": "Jane"}</pre>
<code>requestContext</code>	Un objeto que contiene información adicional sobre la solicitud, como el <code>requestId</code> , el momento de la solicitud y la identidad del intermediario si se autoriza a través de AWS Identity and Access Management (IAM).	
<code>requestContext.accountId</code>	El ID de Cuenta de AWS del propietario de la función.	<code>"123456789012"</code>
<code>requestContext.apiId</code>	El ID de la URL de función.	<code>"33anwqw8fj"</code>
<code>requestContext.authentication</code>	Las URL de funciones no utilizan este parámetro. Lambda establece esto como <code>null</code> .	<code>null</code>

Parámetro	Descripción	Ejemplo
<code>requestContext.authorizer</code>	Un objeto que contiene información sobre la identidad del intermediario, si la URL de función utiliza el tipo de autenticación <code>AWS_IAM</code> . De lo contrario, Lambda establece esto como <code>null</code> .	
<code>requestContext.authorizer.iam.accessKey</code>	La clave de acceso de la identidad del intermediario.	"AKIAIOSFODNN7EXAMPLE"
<code>requestContext.authorizer.iam.accountId</code>	El ID de identidad de Cuenta de AWS del intermediario.	"111122223333"
<code>requestContext.authorizer.iam.callerId</code>	El ID (ID de usuario) del intermediario.	"AIDACKCEVSQ6C2EXAMPLE"
<code>requestContext.authorizer.iam.cognitoIdentity</code>	Las URL de funciones no utilizan este parámetro. Lambda establece esto como <code>null</code> o lo excluye de JSON.	<code>null</code>
<code>requestContext.authorizer.iam.principalOrgId</code>	El ID de la entidad principal de la organización asociado a la identidad del intermediario.	"AIDACKCEVSQORGEXAMPLE"
<code>requestContext.authorizer.iam.userArn</code>	El nombre de recurso de Amazon (ARN) del usuario de la identidad del intermediario.	"arn:aws:iam::111122223333:user/example-user"
<code>requestContext.authorizer.iam.userId</code>	El ID de usuario de la identidad del intermediario.	"AIDACOSFODNN7EXAMPLE2"

Parámetro	Descripción	Ejemplo
<code>requestContext.domainName</code>	El nombre de dominio de la URL de función.	"<url-id>.lambda-url.us-west-2.on.aws"
<code>requestContext.domainPrefix</code>	El prefijo de dominio de la URL de función.	"<url-id>"
<code>requestContext.http</code>	Un objeto que contiene detalles sobre la solicitud HTTP.	
<code>requestContext.http.method</code>	El método HTTP utilizado en esta solicitud. Los valores válidos son GET, POST, PUT, HEAD, OPTIONS, PATCH y DELETE.	GET
<code>requestContext.http.path</code>	Ruta de acceso de la solicitud. Por ejemplo, si la URL de solicitud es <code>https://{url-id}.lambda-url.{region}.on.aws/example/test/demo</code> , el valor de la ruta es <code>/example/test/demo</code> .	<code>/example/test/demo</code>
<code>requestContext.http.protocol</code>	El protocolo de la solicitud.	HTTP/1.1
<code>requestContext.http.sourceIp</code>	La dirección IP de origen de la conexión TCP inmediata que realiza la solicitud.	123.123.123.123

Parámetro	Descripción	Ejemplo
<code>requestContext.http.userAgent</code>	El valor del encabezado de solicitud usuario-agente.	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0
<code>requestContext.requestId</code>	El ID de la solicitud de invocación. Puede utilizar este ID para realizar un seguimiento de los registros de invocación relacionados con la función.	e1506fd5-9e7b-434f-bd42-4f8fa224b599
<code>requestContext.routeKey</code>	Las URL de funciones no utilizan este parámetro. Lambda establece <code>\$default</code> como marcador de posición.	<code>\$default</code>
<code>requestContext.stage</code>	Las URL de funciones no utilizan este parámetro. Lambda establece <code>\$default</code> como marcador de posición.	<code>\$default</code>
<code>requestContext.time</code>	La marca de tiempo de la solicitud.	"07/Sep/2021:22:50:22 +0000"
<code>requestContext.timeEpoch</code>	La marca de tiempo de la solicitud, en fecha de inicio Unix.	"1631055022677"
<code>body</code>	El cuerpo de la solicitud. Si el tipo de contenido de la solicitud es binario, el cuerpo está codificado en base64.	{"key1": "value1", "key2": "value2"}



Parámetro	Descripción	Ejemplo
<code>pathParameters</code>	Las URL de funciones no utilizan este parámetro. Lambda establece esto como <code>null</code> o lo excluye de JSON.	<code>null</code>
<code>isBase64Encoded</code>	TRUE si el cuerpo es una carga binaria y está codificado en base64. FALSE en caso contrario.	FALSE
<code>stageVariables</code>	Las URL de funciones no utilizan este parámetro. Lambda establece esto como <code>null</code> o lo excluye de JSON.	<code>null</code>

## Formato de carga de respuesta

Cuando la función devuelve una respuesta, Lambda analiza la respuesta y la convierte en una respuesta HTTP. Las cargas de respuesta de la función tienen el siguiente formato:

```
{
  "statusCode": 201,
  "headers": {
    "Content-Type": "application/json",
    "My-Custom-Header": "Custom Value"
  },
  "body": "{ \"message\": \"Hello, world!\" }",
  "cookies": [
    "Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",
    "Cookie_2=Value2; Max-Age=78000"
  ],
  "isBase64Encoded": false
}
```

Lambda le infiere el formato de respuesta. Si la función devuelve JSON válido y no devuelve un `statusCode`, Lambda asume lo siguiente:

- `statusCode` es `200`.
- `content-type` es `application/json`.
- `body` es la respuesta de la función.
- `isBase64Encoded` es `false`.

En los ejemplos siguientes se muestra cómo se asigna la salida de la función de Lambda a la carga de respuesta y cómo se asigna la carga de respuesta a la respuesta HTTP final. Cuando el cliente invoca la URL de función, ve la respuesta HTTP.

#### Ejemplo de salida para una respuesta de cadena

Salida de función de Lambda	Salida de respuesta interpretada	Respuesta HTTP (lo que ve el cliente)
<code>"Hello, world!"</code>	<pre>{   "statusCode": 200,   "body": "Hello, world!",   "headers": {     "content-type": "application/json"   },   "isBase64Encoded": false }</pre>	<pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: applicati on/json content-length: 15  "Hello, world!"</pre>

#### Ejemplo de salida para una respuesta JSON

Salida de función de Lambda	Salida de respuesta interpretada	Respuesta HTTP (lo que ve el cliente)
<pre>{   "message": "Hello, world!" }</pre>	<pre>{   "statusCode": 200,   "body": {     "message": "Hello, world!"   } }</pre>	<pre>HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: applicati on/json</pre>

Salida de función de Lambda	Salida de respuesta interpretada	Respuesta HTTP (lo que ve el cliente)
	<pre> }, "headers": {   "content-type": "application/json" }, "isBase64Encoded": false } </pre>	<pre> content-length: 34  {   "message": "Hello, world!" } </pre>

### Ejemplo de salida para una respuesta personalizada

Salida de función de Lambda	Salida de respuesta interpretada	Respuesta HTTP (lo que ve el cliente)
<pre> {   "statusCode": 201,   "headers": {     "Content-Type": "application/json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stri ngify({     "message": "Hello, world!"   }),   "isBase64Encoded": false } </pre>	<pre> {   "statusCode": 201,   "headers": {     "Content-Type": "application/json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stri ngify({     "message": "Hello, world!"   }),   "isBase64Encoded": false } </pre>	<pre> HTTP/2 201 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: applicati on/json content-length: 27 my-custom-header: Custom Value  {   "message": "Hello, world!" } </pre>

## Cookies

Para devolver las cookies de su función, no agregue encabezados `set-cookie` manualmente. En su lugar, incluya las cookies en su objeto de carga de respuesta. Lambda interpreta esto

automáticamente y las agrega como encabezados set-cookie de la respuesta HTTP, como en el siguiente ejemplo.

Salida de función de Lambda	Respuesta HTTP (lo que ve el cliente)
<pre>{   "statusCode": 201,   "headers": {     "Content-Type": "application/ json",     "My-Custom-Header": "Custom Value"   },   "body": JSON.stringify({     "message": "Hello, world!"   }),   "cookies": [     "Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",     "Cookie_2=Value2; Max-Age=7 8000"   ],   "isBase64Encoded": false }</pre>	<pre>HTTP/2 201 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: application/json content-length: 27 my-custom-header: Custom Value set-cookie: Cookie_1=Value2; Expires=21 Oct 2021 07:48 GMT set-cookie: Cookie_2=Value2; Max- Age=78000  {   "message": "Hello, world!" }</pre>

## Supervisión de las URL de funciones de Lambda

Puede utilizar AWS CloudTrail y Amazon CloudWatch para supervisar las URL de funciones.

### Temas

- [Supervisión de las URL de funciones con CloudTrail](#)
- [Métricas de CloudWatch para las URL de funciones](#)

## Supervisión de las URL de funciones con CloudTrail

Para las URL de funciones, Lambda admite automáticamente el registro de las siguientes operaciones de API como eventos en archivos de registros de CloudTrail:

- [CreateFunctionUrlConfig](#)
- [UpdateFunctionUrlConfig](#)
- [DeleteFunctionUrlConfig](#)
- [GetFunctionUrlConfig](#)
- [ListFunctionUrlConfigs](#)

Cada entrada de registro contiene información sobre la identidad del intermediario, cuándo se realizó la solicitud y otros detalles. Puede ver todos los eventos de los últimos 90 días consultando el historial de eventos de CloudTrail. Para retener registros pasados 90 días, puede crear una traza.

De forma predeterminada, CloudTrail no registra solicitudes `InvokeFunctionUrl`, que se consideran eventos de datos. Sin embargo, puede activar el registro de eventos de datos en CloudTrail. Para obtener más información, consulte [Registro de eventos de datos para registros de seguimiento](#) en la Guía del usuario de AWS CloudTrail.

## Métricas de CloudWatch para las URL de funciones

Lambda envía métricas agregadas sobre las solicitudes de URL de funciones a CloudWatch. Con estas métricas, puede supervisar las URL de funciones, crear paneles y configurar alarmas en la consola de CloudWatch.

Las URL de funciones son compatibles con las siguientes métricas de invocación. Recomendamos ver estas métricas con la estadística Sum

- `UrlRequestCount`: el número de solicitudes realizadas a esta URL de función.
- `Url4xxCount`: el número de solicitudes que devolvió un código de estado HTTP 4XX. Los códigos de la serie 4XX indican errores del lado del cliente, como solicitudes erróneas.
- `Url5xxCount`: el número de solicitudes que devolvió un código de estado HTTP 5XX. Los códigos de la serie 5XX indican errores del lado del servidor, como errores de función y tiempos de espera.

Las URL de funciones también son compatibles con la siguiente métrica de rendimiento.

Recomendamos ver esta métrica con las estadísticas `Average` o `Max`.

- `UrlRequestLatency`: el tiempo transcurrido entre el que la URL de función recibe una solicitud y el momento en que la URL de función devuelve una respuesta.

Cada una de estas métricas de invocación y rendimiento es compatible con las siguientes dimensiones:

- `FunctionName`: visualiza métricas agregadas para las URL de las funciones asignadas a la versión `$LATEST` no publicada o a cualquiera de los alias de la función. Por ejemplo, `hello-world-function`.
- `Resource`: visualiza métricas para una URL de función específica. Esto se define mediante el nombre de una función, junto con la versión `$LATEST` no publicada o uno de los alias de la función. Por ejemplo, `hello-world-function:$LATEST`.
- `ExecutedVersion`: visualiza métricas para una URL de función específica en función de la versión ejecutada. Puede utilizar esta dimensión principalmente para realizar un seguimiento de la URL de función asignada a la versión `$LATEST` no publicada.

## Tutorial: Creación de una función de Lambda con una URL de función

En este tutorial, se crea una función de Lambda definida como un archivo .zip con un punto de conexión de la URL de función pública que devuelve el producto de dos números. Para obtener más información acerca de la configuración de las URL de funciones, consulte [URL de funciones](#).

### Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#), para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

#### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.

### Creación de un rol de ejecución

Cree el [rol de ejecución](#) que concederá a su función de Lambda permiso para obtener acceso a los recursos de AWS.

## Para crear un rol de ejecución

1. Abra la página de [Roles](#) de la consola de AWS Identity and Access Management (IAM).
2. Elija Crear rol.
3. En Tipo de entidad de confianza, seleccione Servicio de AWS; a continuación, en Caso de uso, seleccione Lambda.
4. Elija Siguiente.
5. En el panel Políticas de permisos, ingrese **AWSLambdaBasicExecutionRole** en el cuadro de búsqueda.
6. Seleccione la casilla que se encuentra junto a la política administrada por AWS **AWSLambdaBasicExecutionRole** y después seleccione Siguiente.
7. Ingrese **lambda-url-role** en Nombre de rol y, a continuación, seleccione Crear rol.

La política **AWSLambdaBasicExecutionRole** tiene permisos que la función necesita para escribir registros en Registros de Amazon CloudWatch. Más adelante en el tutorial, necesitará el Nombre de recurso de Amazon (ARN) del rol para crear la función de Lambda.

## Para buscar el ARN del rol de ejecución

1. Abra la página de [Roles](#) de la consola de AWS Identity and Access Management (IAM).
2. Seleccione el rol que acaba de crear (**lambda-url-role**).
3. En el panel Resumen, copie el ARN.

## Creación de una función de Lambda con una URL de función (archivo .zip)

Cree una función de Lambda con un punto de conexión de la URL de función mediante un archivo .zip.

### Cómo crear la función

1. Copie el siguiente código de ejemplo en un archivo denominado `index.js`.

#### Example index.js

```
exports.handler = async (event) => {
  let body = JSON.parse(event.body);
  const product = body.num1 * body.num2;
```



```
const response = {
  statusCode: 200,
  body: "The product of " + body.num1 + " and " + body.num2 + " is " +
product,
};
return response;
};
```

2. Cree un paquete de implementación.

```
zip function.zip index.js
```

3. Cree una función de Lambda con el comando `create-function`. Asegúrese de reemplazar el ARN del rol por el ARN de su propio rol de ejecución que copió anteriormente en el tutorial.

```
aws lambda create-function \
  --function-name my-url-function \
  --runtime nodejs18.x \
  --zip-file fileb://function.zip \
  --handler index.handler \
  --role arn:aws:iam::123456789012:role/lambda-url-role
```

4. Agregue una política basada en recursos a la función que otorgue permisos para habilitar el acceso público a la URL de la función.

```
aws lambda add-permission \
  --function-name my-url-function \
  --action lambda:InvokeFunctionUrl \
  --principal "*" \
  --function-url-auth-type "NONE" \
  --statement-id url
```

5. Cree un punto de conexión de la URL para la función con el comando `create-function-url-config`.

```
aws lambda create-function-url-config \
  --function-name my-url-function \
  --auth-type NONE
```

## Prueba del punto de conexión de la URL de función

Invoke la función de Lambda llamando al punto de conexión de la URL de función mediante un cliente HTTP como curl o Postman.

```
curl 'https://abcdefg.lambda-url.us-east-1.on.aws/' \  
-H 'Content-Type: application/json' \  
-d '{"num1": "10", "num2": "10"}'
```

Debería ver los siguientes datos de salida:

```
The product of 10 and 10 is 100
```

## Creación de una función de Lambda con una URL de función (CloudFormation)

También puede crear una función de Lambda con un punto de conexión de la URL de función mediante el tipo de AWS CloudFormation `AWS::Lambda::Url`.

```
Resources:  
  MyUrlFunction:  
    Type: AWS::Lambda::Function  
    Properties:  
      Handler: index.handler  
      Runtime: nodejs18.x  
      Role: arn:aws:iam::123456789012:role/lambda-url-role  
      Code:  
        ZipFile: |  
          exports.handler = async (event) => {  
            let body = JSON.parse(event.body);  
            const product = body.num1 * body.num2;  
            const response = {  
              statusCode: 200,  
              body: "The product of " + body.num1 + " and " + body.num2 + " is " +  
product,  
            };  
            return response;  
          };  
        Description: Create a function with a URL.  
  MyUrlFunctionPermissions:  
    Type: AWS::Lambda::Permission  
    Properties:  
      FunctionName: !Ref MyUrlFunction
```

```
Action: lambda:InvokeFunctionUrl
Principal: "*"
FunctionUrlAuthType: NONE
MyFunctionUrl:
  Type: AWS::Lambda::Url
  Properties:
    TargetFunctionArn: !Ref MyUrlFunction
    AuthType: NONE
```

## Creación de una función de Lambda con una URL de función (AWS SAM)

También puede crear una función de Lambda configurada con una URL de función mediante AWS Serverless Application Model (AWS SAM).

```
ProductFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: function/.
    Handler: index.handler
    Runtime: nodejs18.x
    AutoPublishAlias: live
    FunctionUrlConfig:
      AuthType: NONE
```

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.

2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y seleccione Delete (Eliminar).

# Comprender el escalado de la función de Lambda

La simultaneidad representa la cantidad de solicitudes que la función AWS Lambda puede tolerar al mismo tiempo. Para cada solicitud simultánea, Lambda aprovisiona una instancia independiente del entorno de ejecución. A medida que las funciones reciben más solicitudes, Lambda se encarga automáticamente de escalar la cantidad de entornos de ejecución hasta que alcance el límite de simultaneidad de la cuenta. De forma predeterminada, Lambda proporciona a su cuenta un límite de simultaneidad total de 1000 ejecuciones simultáneas en todas las funciones de una Región de AWS. Para satisfacer las necesidades específicas de la cuenta, puede [solicitar un aumento de la cuota](#) y configurar los controles de simultaneidad en la función para que las funciones críticas no sufran limitaciones.

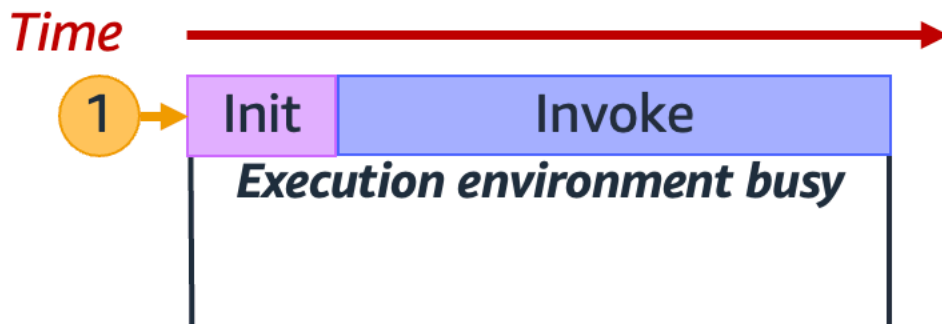
En este tema se explican los conceptos de simultaneidad y el escalado de funciones en Lambda. Al final de este tema, podrá entender cómo calcular la simultaneidad, visualizar las dos opciones principales de control de simultaneidad (reservadas y aprovisionadas), estimar la configuración de control de simultaneidad adecuada y ver las métricas para una mayor optimización.

## Secciones

- [Cómo comprender y visualizar la simultaneidad](#)
- [Calcular la simultaneidad de una función](#)
- [Comprender la simultaneidad reservada y simultaneidad aprovisionada](#)
- [Descripción de la simultaneidad y las solicitudes por segundo](#)
- [Cuotas de simultaneidad](#)
- [Configurar la simultaneidad reservada para una función](#)
- [Configuración de simultaneidad aprovisionada para una función](#)
- [Comportamiento de escalado de Lambda](#)
- [Monitoreo de la simultaneidad](#)

## Cómo comprender y visualizar la simultaneidad

Lambda invoca la función en un [entorno de ejecución](#) seguro y aislado. Para administrar una solicitud, Lambda debe inicializar primero un entorno de ejecución (la [fase de inicio](#)), antes de usarlo para invocar la función (la [fase de invocación](#)):

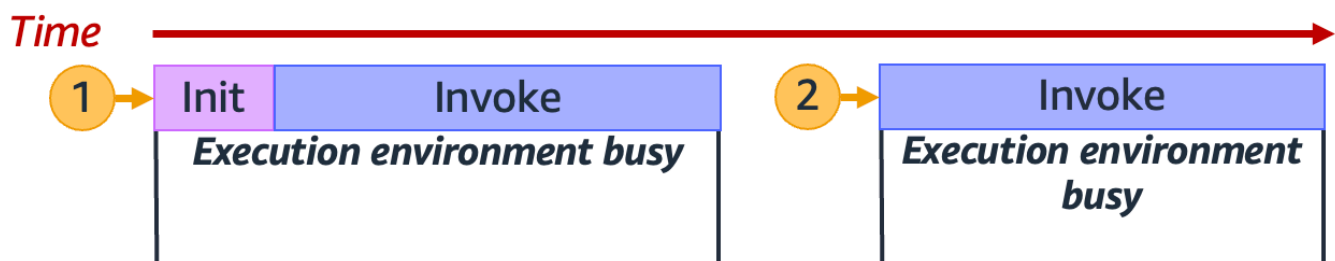


**Note**

Las duraciones reales de inicio e invocación pueden variar en función de muchos factores, como el tiempo de ejecución que elija y el código de la función de Lambda. El diagrama anterior no pretende representar las proporciones exactas de las duraciones de las fases de inicio e invocación.

El diagrama anterior utiliza un rectángulo para representar un único entorno de ejecución. Cuando la función recibe la primera solicitud (representada por un círculo amarillo con la etiqueta 1), Lambda crea un nuevo entorno de ejecución y ejecuta el código fuera del controlador principal durante la fase de inicio. A continuación, Lambda ejecuta el código del controlador principal de la función durante la fase de invocación. Durante todo este proceso, este entorno de ejecución está ocupado y no puede procesar otras solicitudes.

Cuando Lambda termina de procesar la primera solicitud, este entorno de ejecución puede procesar solicitudes adicionales para la misma función. Para las solicitudes posteriores, Lambda no necesita volver a inicializar el entorno.

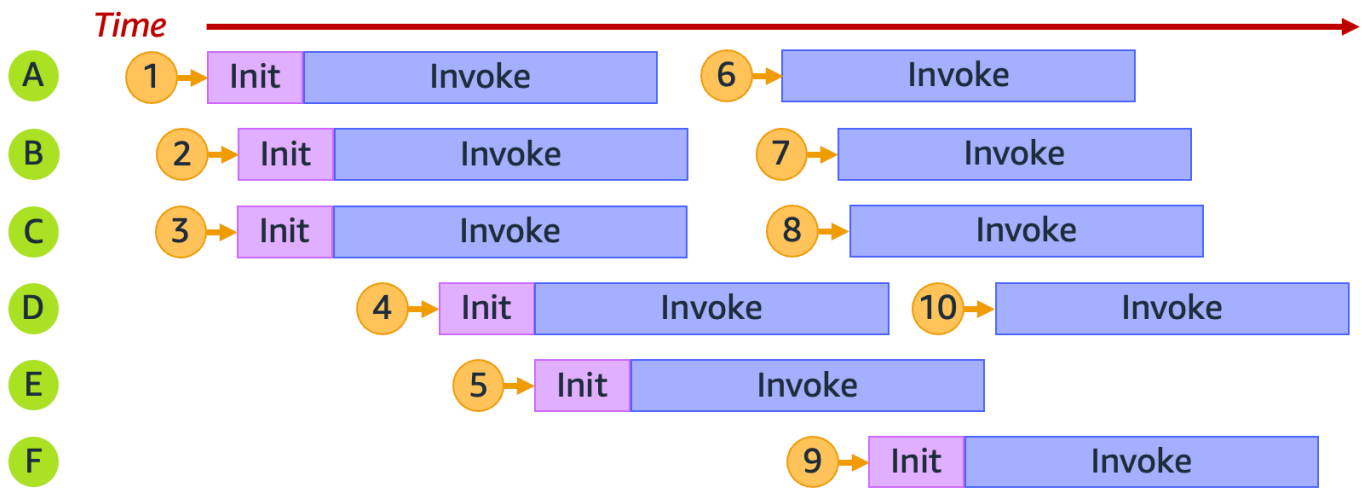


En el diagrama anterior, Lambda reutiliza el entorno de ejecución para administrar la segunda solicitud (representada por el círculo amarillo con la etiqueta 2).

Hasta ahora, nos hemos centrado en una sola instancia del entorno de ejecución (es decir, una simultaneidad de 1). En la práctica, es posible que Lambda necesite aprovisionar varias instancias del entorno de ejecución en paralelo para administrar todas las solicitudes entrantes. Cuando la función recibe una nueva solicitud, puede ocurrir una de estas dos cosas:

- Si una instancia de entorno de ejecución preinicializada está disponible, Lambda la utiliza para procesar la solicitud.
- De lo contrario, Lambda crea una nueva instancia del entorno de ejecución para procesar la solicitud.

Por ejemplo, analicemos lo que ocurre cuando la función recibe 10 solicitudes:



En el diagrama anterior, cada plano horizontal representa una única instancia del entorno de ejecución (etiquetada de la A a F). Así es como Lambda administra cada solicitud:

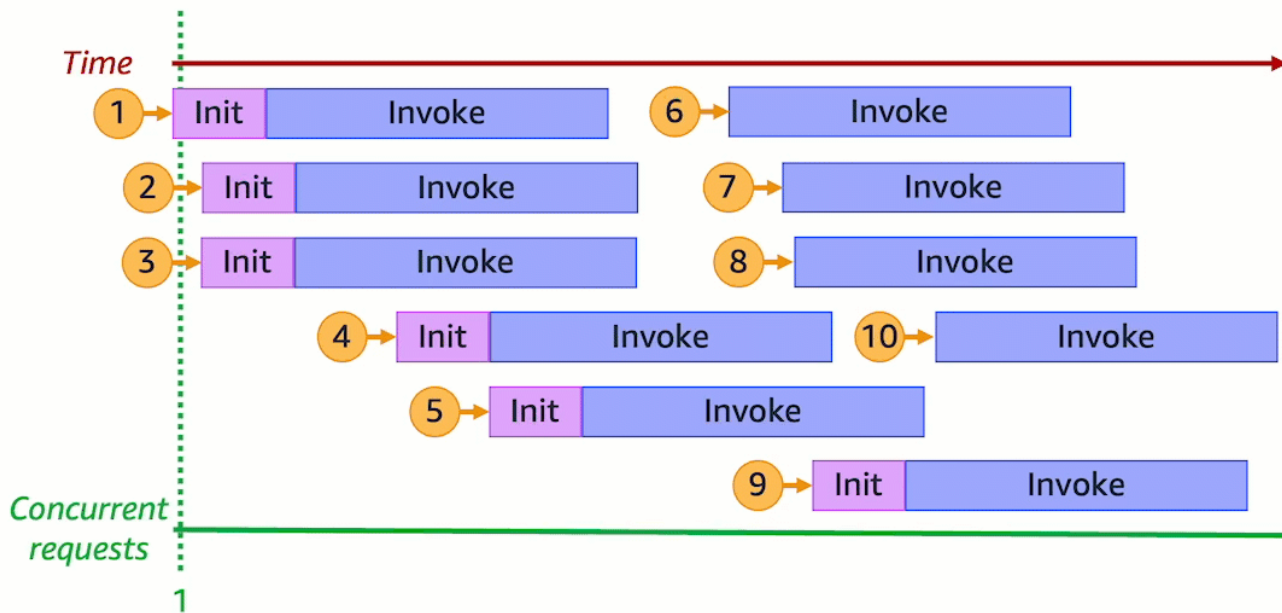
Solicitud	Comportamiento de Lambda	Razonamiento
1	Aprovisiona el nuevo entorno A.	Esta es la primera solicitud; no hay instancias de entorno de ejecución disponibles.

Solicitud	Comportamiento de Lambda	Razonamiento
2	Aprovisiona el nuevo entorno B.	La instancia A del entorno de ejecución existente está ocupada.
3	Aprovisiona el nuevo entorno C.	Las instancias A y B del entorno de ejecución existentes están ocupadas.
4	Aprovisiona el nuevo entorno D.	Las instancias A, B y C del entorno de ejecución existentes están todas ocupadas.
5	Aprovisiona el nuevo entorno E.	Las instancias A, B, C y D del entorno de ejecución existentes están todas ocupadas.
6	Reutiliza el entorno A.	La instancia A del entorno de ejecución terminó de procesar la solicitud 1 y ya está disponible.
7	Reutiliza el entorno B.	La instancia B del entorno de ejecución terminó de procesar la solicitud 2 y ya está disponible.
8	Reutiliza el entorno C.	La instancia C del entorno de ejecución terminó de procesar la solicitud 3 y ya está disponible.
9	Aprovisiona el nuevo entorno F.	Las instancias A, B, C, D y E del entorno de ejecución existentes están todas ocupadas.

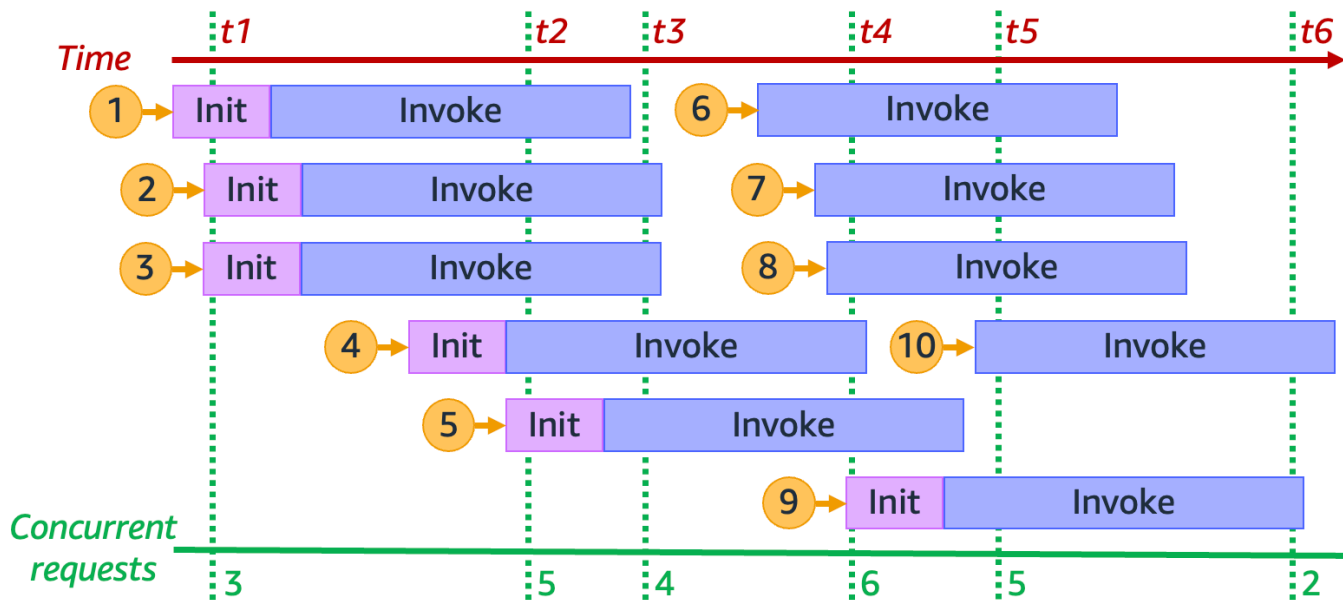


Solicitud	Comportamiento de Lambda	Razonamiento
10	Reutiliza el entorno D.	La instancia D del entorno de ejecución terminó de procesar la solicitud 4 y ya está disponible.

Como respuesta, a medida que la función recibe más solicitudes simultáneas, Lambda escala verticalmente la cantidad de instancias del entorno de ejecución. La siguiente animación registra la cantidad de solicitudes simultáneas a lo largo del tiempo:



Cuando se congela la animación anterior en seis puntos distintos en el tiempo, se obtiene el siguiente diagrama:



En el diagrama anterior, podemos dibujar una línea vertical en cualquier momento y contar la cantidad de entornos que se cruzan con esta línea. Esto nos da la cantidad de solicitudes simultáneas en ese momento. Por ejemplo, en el momento  $t_1$ , existen tres entornos activos que atienden tres solicitudes simultáneas. La cantidad máxima de solicitudes simultáneas de esta simulación se produce en el momento  $t_4$ , cuando hay seis entornos activos que atienden seis solicitudes simultáneas.

En resumen, la simultaneidad de la función es la cantidad de solicitudes simultáneas que se administran a la vez. En respuesta a un aumento en la simultaneidad de la función, Lambda aprovisiona más instancias del entorno de ejecución para satisfacer la demanda de solicitudes.

## Calcular la simultaneidad de una función

En general, la simultaneidad de un sistema es la capacidad de procesar más de una tarea simultáneamente. En Lambda, la simultaneidad es la cantidad de solicitudes en vuelo que la función administra al mismo tiempo. Una forma rápida y práctica de medir la simultaneidad de una función de Lambda consiste en utilizar la siguiente fórmula:

$$\text{Concurrency} = (\text{average requests per second}) * (\text{average request duration in seconds})$$

La simultaneidad se diferencia de las solicitudes por segundo. Por ejemplo, supongamos que la función recibe una media de 100 solicitudes por segundo. Si la duración media de las solicitudes es de un segundo, es cierto que la simultaneidad también es de 100:

$$\text{Concurrency} = (100 \text{ requests/second}) * (1 \text{ second/request}) = 100$$

Sin embargo, si la duración media de la solicitud es de 500 ms, entonces la simultaneidad es de 50:

$$\text{Concurrency} = (100 \text{ requests/second}) * (0.5 \text{ second/request}) = 50$$

¿Qué significa en la práctica una simultaneidad de 50? Si la duración media de las solicitudes es de 500 ms, entonces puede pensar que una instancia de la función puede administrar dos solicitudes por segundo. Luego, se necesitan 50 instancias de la función para administrar una carga de 100 solicitudes por segundo. Una simultaneidad de 50 significa que Lambda debe aprovisionar 50 instancias del entorno de ejecución para administrar esta carga de trabajo de manera eficiente sin ningún tipo de limitación. A continuación, se explica cómo expresar esto en forma de ecuación:

$$\text{Concurrency} = (100 \text{ requests/second}) / (2 \text{ requests/second}) = 50$$

Si la función recibe el doble de solicitudes (200 solicitudes por segundo), pero solo necesita la mitad del tiempo para procesar cada solicitud (250 ms), entonces la simultaneidad sigue siendo de 50:

$$\text{Concurrency} = (200 \text{ requests/second}) * (0.25 \text{ second/request}) = 50$$

## Evaluemos su comprensión de la simultaneidad

Supongamos que tiene una función que tarda, en promedio, 200 ms en ejecutarse. Durante la carga máxima, nota que hay 5000 solicitudes por segundo. ¿Cuál es la simultaneidad de la función durante la carga máxima?

### Respuesta

La duración promedio de la función es de 200 ms o 0,2 segundos. Con la fórmula de simultaneidad, puede introducir los números para obtener una simultaneidad de 1000:

$$\text{Concurrency} = (5,000 \text{ requests/second}) * (0.2 \text{ seconds/request}) = 1,000$$

Como alternativa, una duración promedio de una función de 200 ms significa que la función puede procesar 5 solicitudes por segundo. Para administrar la carga de trabajo de 5000 solicitudes por

segundo, necesita 1000 instancias de entorno de ejecución. Por lo tanto, la simultaneidad es de 1000:

$$\text{Concurrency} = (5,000 \text{ requests/second}) / (5 \text{ requests/second}) = 1,000$$

## Comprender la simultaneidad reservada y simultaneidad aprovisionada

De forma predeterminada, la cuenta tiene un límite de simultaneidad de 1000 ejecuciones simultáneas en todas las funciones de una región. Las funciones comparten este conjunto de 1000 simultaneidades bajo demanda. Si se agota la simultaneidad disponible, la función experimenta limitaciones (es decir, empieza a eliminar solicitudes).

Es posible que algunas de las funciones sean más importantes que otras. Como resultado, es posible que desee configurar los ajustes de simultaneidad para garantizar que las funciones críticas obtengan la simultaneidad que necesitan. Existen dos tipos de controles de simultaneidad disponibles: simultaneidad reservada y simultaneidad aprovisionada.

- Utilice la simultaneidad reservada para reservar una parte de la simultaneidad de la cuenta para una función. Esto resulta útil si no desea que otras funciones ocupen toda la simultaneidad no reservada disponible.
- Utilice la simultaneidad aprovisionada para preinicializar varias instancias de entorno para una función. Esto es útil para reducir las latencias de arranque en frío.

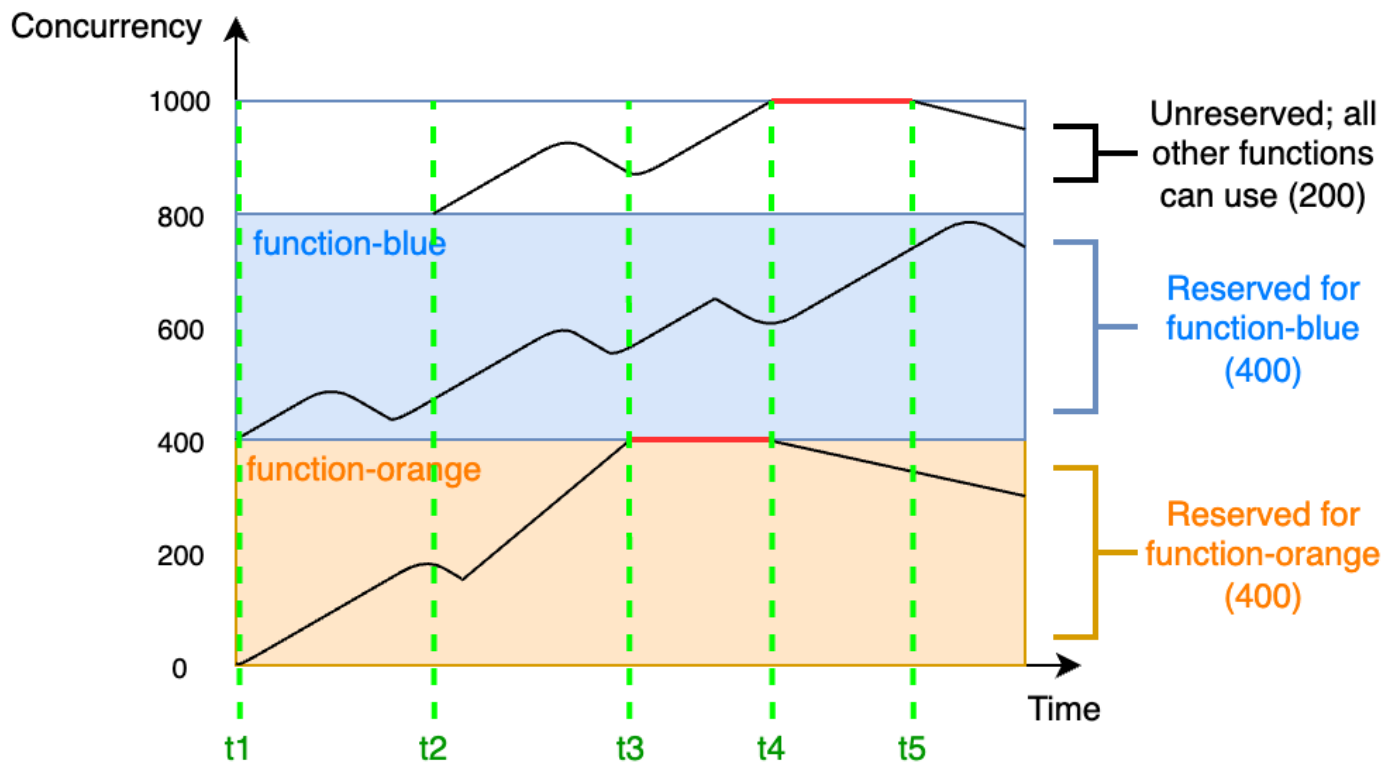
### Simultaneidad reservada

Si desea garantizar que haya una cierta cantidad de simultaneidad disponible para la función en cualquier momento, utilice la simultaneidad reservada.

La simultaneidad reservada es la cantidad máxima de instancias simultáneas que desea asignar a la función. Cuando dedica la simultaneidad reservada a una función, ninguna otra función puede usarla. En otras palabras, la configuración de la simultaneidad reservada puede afectar al grupo de simultaneidad que está disponible para otras funciones. Las funciones que no tienen simultaneidad reservada comparten el conjunto restante de simultaneidad no reservada.

La configuración de la simultaneidad reservada cuenta para el límite general de simultaneidad de la cuenta. No hay ningún cargo por configurar la concurrencia reservada para una función.

Para entender mejor la simultaneidad reservada, tenga en cuenta el siguiente diagrama:



En este diagrama, el límite de simultaneidad de la cuenta para todas las funciones de esta región está en el límite predeterminado de 1000. Supongamos que tiene dos funciones críticas, la `function-blue` y la `function-orange`, que de forma rutinaria se espera que obtengan altos volúmenes de invocación. Decide asignar 400 unidades de simultaneidad reservada a la `function-blue` y 400 unidades de simultaneidad reservada a la `function-orange`. En este ejemplo, todas las demás funciones de la cuenta deben compartir las 200 unidades restantes de simultaneidad no reservada.

El diagrama tiene cinco puntos de interés:

- En el momento t1, tanto la `function-orange` como la `function-blue` comienzan a recibir solicitudes. Cada función comienza a utilizar su parte asignada de las unidades de simultaneidad reservadas.
- En el momento t2, la `function-orange` y la `function-blue` reciben cada vez más solicitudes. Al mismo tiempo, implementa otras funciones de Lambda, que comienzan a recibir solicitudes. No asigna la simultaneidad reservada a estas otras funciones. Estas empiezan a utilizar las 200 unidades restantes de simultaneidad no reservada.

- En el momento  $t_3$ , la `function-orange` alcanza la simultaneidad máxima de 400. Aunque hay simultaneidad no utilizada en otras partes de la cuenta, la `function-orange` no puede acceder a esta. La línea roja indica que la `function-orange` se está sufriendo limitaciones y Lambda podría anular las solicitudes.
- En el momento  $t_4$ , la `function-orange` comienza a recibir menos solicitudes y ya no tiene limitaciones. Sin embargo, las otras funciones experimentan un aumento en el tráfico y comienzan a sufrir limitaciones. Si bien hay simultaneidad no utilizada en otras partes de la cuenta, estas otras funciones no pueden acceder a esta. La línea roja indica que las demás funciones están sufriendo limitaciones.
- En el momento  $t_5$ , otras funciones comienzan a recibir menos solicitudes y ya no experimentan limitaciones.

En este ejemplo, observe que reservar la simultaneidad tiene los siguientes efectos:

- La función se puede escalar independientemente de otras funciones de la cuenta. Todas las funciones de la cuenta en la misma región que no tienen simultaneidad reservada comparten el grupo de simultaneidad no reservada. Sin simultaneidad reservada, otras funciones pueden llegar a utilizar toda la simultaneidad disponible. Esto impide que las funciones críticas se escalen verticalmente cuando sea necesario.
- La función no se puede escalar horizontalmente de forma descontrolada. La simultaneidad reservada limita a la simultaneidad máxima de la función. Esto significa que la función no puede usar la simultaneidad reservada para otras funciones ni la simultaneidad del grupo no reservado. Puede reservar simultaneidad para evitar que la función utilice toda la simultaneidad disponible en la cuenta o que sobrecargue los recursos empleados posteriormente.
- Es posible que no pueda utilizar toda la simultaneidad disponible en la cuenta. Reservar la simultaneidad cuenta para el límite de simultaneidad de la cuenta, pero esto también significa que otras funciones no pueden usar esa parte de la simultaneidad reservada. Si la función no consume toda la simultaneidad que reserva para esta, efectivamente está desperdiciando esa simultaneidad. Esto no es un problema, a menos que otras funciones de la cuenta puedan aprovechar la simultaneidad desperdiciada.

Para saber cómo administrar la configuración de la simultaneidad reservada para las funciones, consulte [Configurar la simultaneidad reservada para una función](#).

## Simultaneidad aprovisionada

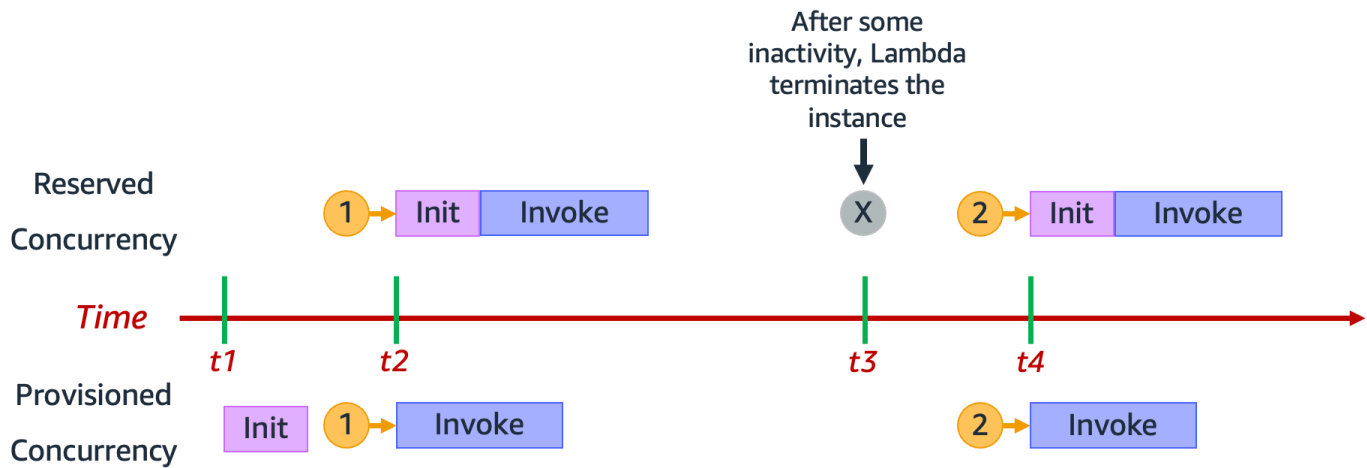
La simultaneidad reservada se utiliza para definir la cantidad máxima de entornos de ejecución reservados para una función de Lambda. Sin embargo, ninguno de estos entornos viene preinicializado. Como resultado, las invocaciones de la función pueden tardar más porque Lambda primero debe inicializar el nuevo entorno antes de poder usarlo para invocar la función. Cuando Lambda tiene que inicializar un nuevo entorno para llevar a cabo una invocación, esto se denomina arranque en frío. Para mitigar los arranques en frío, puede utilizar la simultaneidad aprovisionada.

La simultaneidad aprovisionada es la cantidad de entornos de ejecución preinicializados que desea asignar a la función. Si se establece la simultaneidad aprovisionada en una función, Lambda inicializa esa cantidad de entornos de ejecución para que estén preparados para responder a las solicitudes de la función.

### Note

Utilizar la simultaneidad aprovisionada genera cargos en su cuenta. Si trabaja con los entornos de ejecución de Java 11 o Java 17, también puede utilizar Lambda SnapStart para mitigar los problemas de arranque en frío sin costo adicional. SnapStart utiliza instantáneas almacenadas en caché de su entorno de ejecución para mejorar de forma significativa el rendimiento de inicio. No puede utilizar SnapStart y la simultaneidad aprovisionada en la misma versión de la función. Para obtener más información sobre las características, las limitaciones y las regiones admitidas de SnapStart, consulte [Mejora del rendimiento de inicio con Lambda SnapStart](#).

Cuando se utiliza la simultaneidad aprovisionada, Lambda sigue reciclando los entornos de ejecución en segundo plano. Sin embargo, Lambda se asegura, en todos los casos y en cualquier momento, de que la cantidad de entornos preinicializados sea igual al valor de la configuración de simultaneidad aprovisionada por la función. Este comportamiento difiere de la simultaneidad reservada, en la que Lambda puede terminar por completo un entorno tras un periodo de inactividad. En el siguiente diagrama se demuestra esto, comparando el ciclo de vida de un único entorno de ejecución cuando se configura la función mediante la simultaneidad reservada, en contraposición a la simultaneidad aprovisionada.

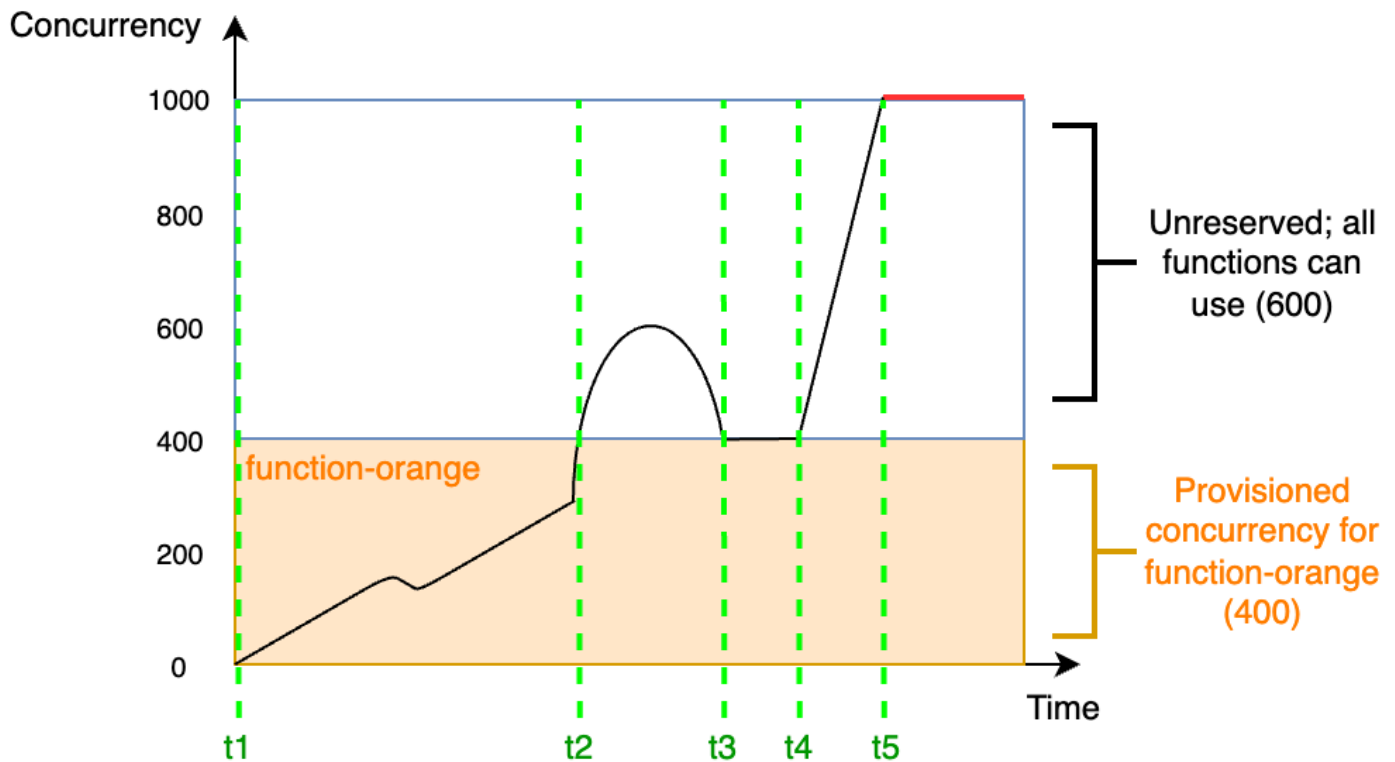


El diagrama tiene cuatro puntos de interés:

Tiempo	Simultaneidad reservada	Simultaneidad aprovisionada
t1	No ocurre nada.	Lambda preinicializa una instancia del entorno de ejecución.
t2	Se presenta la solicitud 1. Lambda debe inicializar una nueva instancia del entorno de ejecución.	Se presenta la solicitud 1. Lambda utiliza la instancia de entorno preinicializada.
t3	Tras un tiempo de inactividad, Lambda termina la instancia del entorno activo.	No ocurre nada.
t4	Se presenta la solicitud 2. Lambda debe inicializar una nueva instancia del entorno de ejecución.	Se presenta la solicitud 2. Lambda utiliza la instancia de entorno preinicializada.

Para comprender mejor la simultaneidad aprovisionada, tenga en cuenta el siguiente diagrama:





En este diagrama, tiene un límite de simultaneidad de cuentas de 1000. Decide otorgar 400 unidades de simultaneidad aprovisionadas a la `function-orange`. Todas las funciones de la cuenta, incluida la `function-orange`, pueden usar las 600 unidades restantes de simultaneidad no reservadas.

El diagrama tiene cinco puntos de interés:

- En el momento `t1`, la `function-orange` comienza a recibir solicitudes. Dado que Lambda ha preinicializado 400 instancias del entorno de ejecución, la `function-orange` está lista para la invocación inmediata.
- En el momento `t2`, la `function-orange` alcanza las 400 solicitudes simultáneas. Como resultado, la `function-orange` agota la simultaneidad aprovisionada. Sin embargo, dado que todavía hay simultaneidad no reservada disponible, Lambda puede usarla para administrar solicitudes adicionales a la `function-orange` (sin limitaciones). Lambda debe crear nuevas instancias para atender estas solicitudes, y es posible que la función experimente latencias de arranque en frío.
- En el momento `t3`, la `function-orange` vuelve a las 400 solicitudes simultáneas tras un breve aumento en el tráfico. Nuevamente, Lambda puede administrar todas las solicitudes sin latencias de arranque en frío.



- En el momento  $t_1$ , la `function-orange` comienza a recibir solicitudes. Dado que Lambda ha preinicializado 200 instancias del entorno de ejecución, la `function-orange` está lista para la invocación inmediata.
- En el momento  $t_2$ , la `function-orange` utiliza toda su simultaneidad aprovisionada. La `function-orange` puede seguir atendiendo solicitudes mediante la simultaneidad reservada, pero estas solicitudes pueden experimentar latencias de arranque en frío.
- En el momento  $t_3$ , la `function-orange` alcanza las 400 solicitudes simultáneas. Como resultado, la `function-orange` agota toda su simultaneidad reservada. Como la `function-orange` no puede utilizar la simultaneidad no reservada, las solicitudes comienzan a sufrir limitaciones.
- En el momento  $t_4$ , la `function-orange` comienza a recibir menos solicitudes y ya no experimenta limitaciones.
- En el momento  $t_5$ , la `function-orange` se reduce a 200 solicitudes simultáneas, por lo que todas las solicitudes pueden volver a utilizar la simultaneidad aprovisionada (es decir, sin latencias de arranque en frío).

Tanto la simultaneidad reservada como la aprovisionada se tienen en cuenta para el límite de simultaneidad de la cuenta y [las cuotas regionales](#). En otras palabras, la simultaneidad reservada y aprovisionada puede afectar al grupo de simultaneidad que está disponible para otras funciones. La configuración de la simultaneidad aprovisionada genera cargos para su Cuenta de AWS.

#### Note

Si el volumen de simultaneidad aprovisionada en las versiones y alias de una función se suma a la simultaneidad reservada de la función, entonces todas las invocaciones se ejecutan en la simultaneidad aprovisionada. Esta configuración también tiene el efecto de aplicar una limitación controlada a la versión sin publicar de la función (`$LATEST`), lo que impide que se ejecute. No puede asignar más concurrencia aprovisionada que la concurrencia reservada para una función.

Para administrar la configuración de la simultaneidad aprovisionada para las funciones, consulte [Configuración de simultaneidad aprovisionada para una función](#). Para automatizar el escalado de simultaneidad aprovisionada en función de un cronograma o la utilización de la aplicación, consulte [Uso de Application Auto Scaling para automatizar la administración de simultaneidad aprovisionada](#).

## Cómo asigna Lambda la simultaneidad aprovisionada

La simultaneidad aprovisionada no se pone en línea inmediatamente después de configurarla. Lambda comienza a asignar simultaneidad aprovisionada después de uno o dos minutos de preparación. Para cada función, Lambda puede aprovisionar hasta 6000 entornos de ejecución por minuto, independientemente de su Región de AWS. Es exactamente igual a la [tasa de escalado de simultaneidad](#) de las funciones.

Cuando envíe una solicitud para asignar la simultaneidad aprovisionada, no podrá acceder a ninguno de esos entornos hasta que Lambda termine de asignarlos por completo. Por ejemplo, si solicita 5000 entornos de simultaneidad aprovisionada, ninguna de sus solicitudes puede utilizar la simultaneidad aprovisionada hasta que Lambda termine de asignar por completo los 5000 entornos de ejecución.

## Comparación de la simultaneidad reservada con la aprovisionada

En la siguiente tabla se resumen y se comparan la simultaneidad reservada y la aprovisionada.

Tema	Simultaneidad reservada	Simultaneidad aprovisionada
Definición	Cantidad máxima de instancias del entorno de ejecución para la función.	Cantidad fija de instancias del entorno de ejecución preaprovisionadas para la función.
Comportamiento de aprovisionamiento	Lambda aprovisiona nuevas instancias bajo demanda.	Lambda aprovisiona previamente las instancias (es decir, antes de que la función comience a recibir solicitudes).
Comportamiento de arranque en frío	Es posible la latencia de arranque en frío, ya que Lambda debe crear nuevas instancias bajo demanda.	La latencia de arranque en frío no es posible, ya que Lambda no necesita crear instancias bajo demanda.
Comportamiento de las limitaciones	La función experimenta limitaciones cuando se alcanza el límite de simultaneidad reservado.	Si la simultaneidad reservada no está configurada, la función utiliza la simultaneidad no reservada cuando se alcanza

Tema	Simultaneidad reservada	Simultaneidad aprovisionada
El comportamiento predeterminado no está configurado	La función utiliza la simultaneidad no reservada disponible en la cuenta.	<p>Lambda no preaprovisiona ninguna instancia. En su lugar, si la simultaneidad reservada no está configurada, la función utiliza la simultaneidad no reservada disponible en la cuenta.</p> <p>Si se configura la simultaneidad reservada, la función usa la simultaneidad reservada.</p>
Precios	Sin cargo adicional.	Conlleva cargos adicionales.

## Descripción de la simultaneidad y las solicitudes por segundo

Como se mencionó en la sección anterior, la simultaneidad se diferencia de las solicitudes por segundo. Esta es una distinción importante cuando se trabaja con funciones que tienen una duración promedio de solicitud inferior a 100 ms.

En todas las funciones de su cuenta, Lambda aplica un límite de solicitudes por segundo igual a 10 veces la simultaneidad de la cuenta. Por ejemplo, dado que el límite predeterminado de simultaneidad de la cuenta es de 1000, las funciones de su cuenta pueden gestionar un máximo de 10 000 solicitudes por segundo.

Por ejemplo, considere una función con una duración promedio de solicitud de 50 ms. Con 20 000 solicitudes por segundo, la simultaneidad de esta función es la siguiente:

$$\text{Concurrency} = (20,000 \text{ requests/second}) * (0.05 \text{ second/request}) = 1,000$$

Según este resultado, se podría esperar que el límite de simultaneidad de la cuenta de 1000 sea suficiente para gestionar esta carga. Sin embargo, debido al límite de 10 000 solicitudes por segundo, la función solo puede gestionar 10 000 solicitudes por segundo del total de 20 000. Esta función experimenta una limitación.

La lección es que debe tener en cuenta tanto la simultaneidad como las solicitudes por segundo al configurar los ajustes de simultaneidad de las funciones. En este caso, debe solicitar un aumento del límite de simultaneidad de la cuenta a 2000, ya que esto aumentaría el límite total de solicitudes por segundo a 20 000.

#### Note

En función de este límite de solicitudes por segundo, es incorrecto decir que cada entorno de ejecución de Lambda solo puede gestionar un máximo de 10 solicitudes por segundo. En lugar de observar la carga en cualquier entorno de ejecución individual, Lambda solo tiene en cuenta la simultaneidad general y las solicitudes totales por segundo al momento de calcular las cuotas.

Ponga a prueba su comprensión de la simultaneidad (funciones de menos de 100 ms)

Supongamos que tiene una función que tarda, en promedio, 20 ms en ejecutarse. Durante la carga máxima, nota que hay 30 000 solicitudes por segundo. ¿Cuál es la simultaneidad de la función durante la carga máxima?

Respuesta

La duración promedio de la función es de 20 ms o 0,02 segundos. Con la fórmula de simultaneidad, puede introducir los números para obtener una simultaneidad de 600:

$$\text{Concurrency} = (30,000 \text{ requests/second}) * (0.02 \text{ seconds/request}) = 600$$

De forma predeterminada, el límite de simultaneidad de la cuenta de 1000 parece ser suficiente para gestionar esta carga. Sin embargo, el límite de 10 000 solicitudes por segundo no es suficiente para gestionar las 30 000 solicitudes entrantes por segundo. Para admitir por completo las 30 000 solicitudes, debe solicitar un aumento del límite de simultaneidad de la cuenta a 3000 o más.

El límite de solicitudes por segundo se aplica a todas las cuotas en Lambda que implican simultaneidad. En otras palabras, se aplica a las funciones sincrónicas bajo demanda, las funciones que utilizan la simultaneidad aprovisionada y [al comportamiento de escalado de la simultaneidad](#). Por ejemplo, a continuación, se muestran algunos escenarios en los que debe considerar con detenimiento sus límites de simultaneidad y de solicitudes por segundo:

- Una función que utilice la simultaneidad bajo demanda puede experimentar un aumento de ráfaga de simultaneidad de 500 cada 10 segundos o de 5000 solicitudes por segundo cada 10 segundos, lo que ocurra primero.
- Supongamos que tiene una función que tiene una asignación de simultaneidad aprovisionada de 10. Esta función se extiende a la simultaneidad bajo demanda después de una simultaneidad de 10 o de 100 solicitudes por segundo, lo que ocurra primero.

## Cuotas de simultaneidad

Lambda establece cuotas para la cantidad total de simultaneidad que puede utilizar en todas las funciones de una región. Estas cuotas existen en dos niveles:

- A nivel de cuenta, de forma predeterminada, las funciones pueden tener hasta 1000 unidades de simultaneidad. Para solicitar un aumento de cuota, consulte [Solicitud de aumento de cuota](#) en la Guía del usuario de Service Quotas.
- A nivel de función, de forma predeterminada, puede reservar hasta 900 unidades de simultaneidad en todas las funciones de forma predeterminada. Independientemente del límite total de simultaneidad de la cuenta, Lambda siempre reserva 100 unidades de simultaneidad para las funciones que no reserven explícitamente la simultaneidad. Por ejemplo, si aumentó el límite de simultaneidad de la cuenta a 2000, entonces puede reservar hasta 1900 unidades de simultaneidad a nivel de función.
- Tanto a nivel de la cuenta como de la función, Lambda también impone un límite de solicitudes por segundo igual a 10 veces la cuota de simultaneidad correspondiente. Por ejemplo, esto se aplica a la simultaneidad a nivel de la cuenta, a las funciones que utilizan la simultaneidad bajo demanda, a las funciones que utilizan la simultaneidad aprovisionada y [al comportamiento de escalado de la simultaneidad](#). Para obtener más información, consulte [the section called “Descripción de la simultaneidad y las solicitudes por segundo”](#).

Para comprobar la cuota de simultaneidad actual a nivel de cuenta, utilice la AWS Command Line Interface (AWS CLI) para ejecutar el siguiente comando:

```
aws lambda get-account-settings
```

Debería ver una salida con un aspecto similar al siguiente:

```
{
  "AccountLimit": {
    "TotalCodeSize": 80530636800,
    "CodeSizeUnzipped": 262144000,
    "CodeSizeZipped": 52428800,
    "ConcurrentExecutions": 1000,
    "UnreservedConcurrentExecutions": 900
  },
  "AccountUsage": {
    "TotalCodeSize": 410759889,
    "FunctionCount": 8
  }
}
```

`ConcurrentExecutions` es la cuota total de simultaneidad a nivel de cuenta.

`UnreservedConcurrentExecutions` es la cantidad de simultaneidad reservada que aún puede asignar a sus funciones.

A medida que una función recibe más solicitudes, Lambda escala de forma automática la cantidad de entornos de ejecución para gestionar estas solicitudes hasta que su cuenta alcance la cuota de simultaneidad. Sin embargo, para protegerse contra el exceso de escalado en respuesta a ráfagas repentinas de tráfico, Lambda limita la rapidez con la que sus funciones pueden escalar. Esta tasa de escalado de simultaneidad es la tasa máxima a la que las funciones de su cuenta pueden escalar en respuesta a un aumento de las solicitudes. (Es decir, la rapidez con la que Lambda puede crear nuevos entornos de ejecución). La tasa de escalado de simultaneidad difiere del límite de simultaneidad de la cuenta, que es la cantidad total de simultaneidad disponible para sus funciones.

En cada Región de AWS y para cada función, su tasa de escalado de simultaneidad es de 1000 instancias del entorno de ejecución cada 10 segundos (o 10 000 solicitudes por segundo cada 10 segundos). En otras palabras, cada 10 segundos, Lambda puede asignar como máximo 1000 instancias de entorno de ejecución adicionales o admitir 10 000 solicitudes por segundo adicionales a cada una de sus funciones.

Por lo general, no debe preocuparse por esta limitación. La tasa de escalado de Lambda es suficiente para la mayoría de los casos de uso.



Es importante destacar que la tasa de escalado de simultaneidad es un límite de función. Esto significa que cada función de su cuenta puede escalar independientemente de otras funciones.

Para obtener más información acerca de comportamientos de escalado, consulte [Comportamiento de escalado de Lambda](#).

# Configurar la simultaneidad reservada para una función

En Lambda, la [simultaneidad](#) es la cantidad de solicitudes en tránsito que la función administra en la actualidad. Hay dos tipos de controles de concurrencia disponibles:

- **Simultaneidad reservada:** representa la cantidad máxima de instancias simultáneas asignadas a la función. Cuando una función ha reservado simultaneidad, ninguna otra función puede usarla. La simultaneidad reservada es útil para garantizar que las funciones más importantes siempre tengan suficiente simultaneidad para manejar las solicitudes entrantes. No hay ningún cargo adicional por configurar la concurrencia reservada para una función.
- **Simultaneidad aprovisionada:** es la cantidad de entornos de ejecución preinicializados asignados a la función. Estos entornos de ejecución están listos para responder de forma inmediata a las solicitudes de funciones entrantes. La simultaneidad aprovisionada es útil para reducir las latencias de arranque en frío de las funciones. La configuración de la simultaneidad aprovisionada genera cargos adicionales para su Cuenta de AWS.

En este tema se detalla cómo administrar y configurar la simultaneidad reservada. Para obtener una descripción general conceptual de estos dos tipos de controles de simultaneidad, consulte [Simultaneidad reservada y simultaneidad aprovisionada](#). Para obtener información sobre la configuración de la simultaneidad aprovisionada, consulte [the section called “Configuración de simultaneidad aprovisionada”](#).

## Note

Las funciones de Lambda enlazadas a una asignación de orígenes de eventos de Amazon MQ tienen una simultaneidad máxima predeterminada. Para Apache Active MQ, el número máximo de instancias simultáneas es 5. Para Rabbit MQ, el número máximo de instancias simultáneas es 1. Establecer una simultaneidad reservada o aprovisionada para su función no cambia estos límites. Para solicitar un aumento de la simultaneidad máxima predeterminada al utilizar Amazon MQ, póngase en contacto con AWS Support.

## Secciones

- [Configuración de la simultaneidad reservada](#)
- [Estimar con precisión la simultaneidad reservada requerida para una función](#)

## Configuración de la simultaneidad reservada

Puede configurar la simultaneidad reservada de una función a través la consola o la API de Lambda.

Para reservar la simultaneidad de una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función para la que desea reservar la simultaneidad.
3. Elija Configuración y, a continuación, elija Simultánea.
4. En Simultaneidad, elija Editar.
5. Seleccione Simultaneidad de reserva. Escriba la cantidad de simultaneidad que reservar para la función.
6. Seleccione Guardar.

Puede reservar hasta el valor de Simultaneidad de cuenta sin reserva menos 100. Las 100 unidades restantes son para funciones que no utilizan la simultaneidad reservada. Por ejemplo, si la cuenta tiene un límite de simultaneidad de 1000, no puede reservar las 1000 unidades de simultaneidad para una sola función.


### Edit concurrency

#### Concurrency

Unreserved account concurrency: 0

Use unreserved account concurrency

Reserve concurrency

 The unreserved account concurrency can't go below 100.

Cancel **Save**

Reservar simultaneidad para una función puede afectar al grupo de simultaneidad que está disponible para otras funciones. Por ejemplo, si reserva 100 unidades de simultaneidad

para `function-a`, otras funciones de su cuenta deberán compartir las 900 unidades restantes, incluso si `function-a` no utiliza las 100 unidades de simultaneidad reservadas.

Para aplicar la limitación controlada a una función de forma intencional, establezca la simultaneidad reservada en 0. Esto impide que la función pueda procesar los eventos hasta que elimine el límite.

Para configurar la simultaneidad reservada con la API de Lambda, use las siguientes operaciones de la API.

- [PutFunctionConcurrency](#)
- [GetFunctionConcurrency](#)
- [DeleteFunctionConcurrency](#)

Para configurar la simultaneidad reservada con la AWS Command Line Interface (CLI), use el comando `put-function-concurrency`. El siguiente comando reserva 100 unidades de simultaneidad para una función llamada `my-function`:

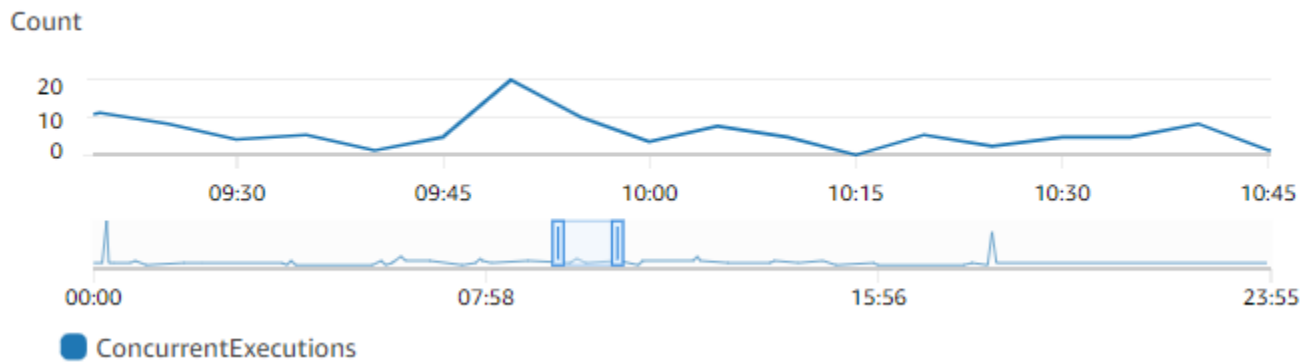
```
aws lambda put-function-concurrency --function-name my-function \  
--reserved-concurrent-executions 100
```

Debería ver una salida con un aspecto similar al siguiente:

```
{  
  "ReservedConcurrentExecutions": 100  
}
```

## Estimar con precisión la simultaneidad reservada requerida para una función

Si actualmente la función atiende el tráfico, puede ver fácilmente sus métricas de simultaneidad mediante las [métricas de CloudWatch](#). En concreto, la métrica `ConcurrentExecutions` muestra la cantidad de invocaciones simultáneas para cada función de la cuenta.



Según el gráfico anterior, esta función atiende un promedio de 5 a 10 solicitudes simultáneas en cualquier momento y alcanza un máximo de 20 solicitudes en un día normal. Supongamos que hay muchas otras funciones en la cuenta. Si esta función es esencial para la aplicación y no quiere descartar ninguna solicitud, utilice un número igual o mayor a 20 como configuración de la simultaneidad reservada.

Como alternativa, recuerde que también puede [calcular la simultaneidad](#) mediante la siguiente fórmula:

$$\text{Concurrency} = (\text{average requests per second}) * (\text{average request duration in seconds})$$

Al multiplicar el promedio de solicitudes por segundo por la duración promedio de las solicitudes en segundos, obtendrá una estimación aproximada de la cantidad de simultaneidad que necesita reservar. Puede estimar el promedio de solicitudes por segundo mediante la métrica de `Invocation` y la duración promedio de las solicitudes en segundos mediante la métrica de `Duration`. Consulte [Ver las métricas de funciones de Lambda](#) para obtener más detalles.

También debería familiarizarse con las limitaciones de rendimiento ascendentes y descendentes. Si bien las funciones de Lambda se escalan sin problemas con la carga, es posible que las dependencias ascendentes y descendentes no tengan las mismas capacidades de rendimiento. Si necesita limitar cuán alto se puede escalar su función, configure la simultaneidad reservada de su función.

# Configuración de simultaneidad aprovisionada para una función

En Lambda, la [simultaneidad](#) es la cantidad de solicitudes en tránsito que la función administra en la actualidad. Hay dos tipos de controles de concurrencia disponibles:

- **Simultaneidad reservada:** representa la cantidad máxima de instancias simultáneas asignadas a la función. Cuando una función ha reservado simultaneidad, ninguna otra función puede usarla. La simultaneidad reservada es útil para garantizar que las funciones más importantes siempre tengan suficiente simultaneidad para manejar las solicitudes entrantes. No hay ningún cargo adicional por configurar la concurrencia reservada para una función.
- **Simultaneidad aprovisionada:** es la cantidad de entornos de ejecución preinicializados asignados a la función. Estos entornos de ejecución están listos para responder de forma inmediata a las solicitudes de funciones entrantes. La simultaneidad aprovisionada es útil para reducir las latencias de arranque en frío de las funciones. La configuración de la simultaneidad aprovisionada genera cargos adicionales para su Cuenta de AWS.

En este tema se detalla cómo administrar y configurar la simultaneidad aprovisionada. Para obtener una descripción general conceptual de estos dos tipos de controles de simultaneidad, consulte [Simultaneidad reservada y simultaneidad aprovisionada](#). Para obtener más información sobre los límites de simultaneidad reservada, consulte [the section called “Configuración de la simultaneidad reservada”](#).

## Note

Las funciones de Lambda enlazadas a una asignación de orígenes de eventos de Amazon MQ tienen una simultaneidad máxima predeterminada. Para Apache Active MQ, el número máximo de instancias simultáneas es 5. Para Rabbit MQ, el número máximo de instancias simultáneas es 1. Establecer una simultaneidad reservada o aprovisionada para su función no cambia estos límites. Para solicitar un aumento de la simultaneidad máxima predeterminada al utilizar Amazon MQ, póngase en contacto con AWS Support.

## Secciones

- [Configuración de simultaneidad aprovisionada](#)
- [Estimar con precisión la simultaneidad aprovisionada requerida para una función](#)
- [Optimizar el código de la función cuando se utiliza la simultaneidad aprovisionada](#)

- [Usar variables de entorno para ver y controlar el comportamiento de simultaneidad aprovisionado](#)
- [Comprenda el comportamiento de registro y facturación con la simultaneidad aprovisionada](#)
- [Uso de Application Auto Scaling para automatizar la administración de simultaneidad aprovisionada](#)

## Configuración de simultaneidad aprovisionada

Puede configurar los ajustes de la simultaneidad aprovisionada para una función mediante la consola o la API de Lambda.

Para asignar la simultaneidad aprovisionada para una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función para la que desee asignar la simultaneidad aprovisionada.
3. Elija Configuración y, a continuación, elija Simultánea.
4. En Configuraciones de concurrencia aprovisionadas, seleccione Agregar configuración.
5. Elija el tipo de calificador y el alias o la versión.

### Note

No puede utilizar la simultaneidad aprovisionada con la versión \$LATEST de ninguna función.

Si su función tiene un origen de eventos, asegúrese de que este apunte al alias o la versión correctos de la función. De lo contrario, la función no utilizará los entornos de simultaneidad aprovisionada.

6. Ingrese un número en Simultaneidad aprovisionada. Lambda proporciona una estimación de los costos mensuales.
7. Seleccione Guardar.

Puede configurar hasta el valor de Simultaneidad de cuenta sin reserva de su cuenta, menos 100. Las 100 unidades restantes son para funciones que no utilizan la simultaneidad reservada. Por ejemplo, si su cuenta tiene un límite de simultaneidad de 1000 y no ha asignado ninguna simultaneidad reservada o aprovisionada a ninguna de sus otras funciones, puede configurar un máximo de 900 unidades de simultaneidad aprovisionadas para una sola función.

### Provisioned concurrency

To enable your function to scale without fluctuations in latency, use provisioned concurrency. You can use Application Auto Scaling to automatically adjust provisioned concurrency to maintain a configured target utilization. Provisioned concurrency runs continually and has separate pricing for concurrency and execution duration. [Learn more](#)

**\$0.00 per month in addition to pricing for duration and requests.** [Pricing](#)

**⚠ The maximum allowed provisioned concurrency is 900, based on the unreserved concurrency available (1000) minus the minimum unreserved account concurrency (100).**

900 available

**⊗ Please correct the errors above.**

La configuración de la simultaneidad aprovisionada de una función tiene un impacto en el grupo de simultaneidad que está disponible para otras funciones. Por ejemplo, si configura 100 unidades de simultaneidad aprovisionadas para `function-a`, otras funciones de su cuenta deberán compartir las 900 unidades de simultaneidad restantes. Esto es válido incluso si `function-a` no utiliza las 100 unidades.

Es posible asignar simultaneidad reservada y simultaneidad aprovisionada para la misma función. En esos casos, la simultaneidad aprovisionada no puede superar la reservada.

Esta limitación se extiende a las versiones de funciones. La simultaneidad aprovisionada máxima que puede asignar a una versión de función específica es igual a la simultaneidad reservada de la función menos la simultaneidad aprovisionada en otras versiones de la función.

Para configurar la simultaneidad aprovisionada con la API de Lambda, use las siguientes operaciones de la API.

- [PutProvisionedConcurrencyConfig](#)
- [GetProvisionedConcurrencyConfig](#)
- [ListProvisionedConcurrencyConfigs](#)
- [DeleteProvisionedConcurrencyConfig](#)

Por ejemplo, para configurar la simultaneidad aprovisionada con la AWS Command Line Interface (CLI), utilice el comando `put-provisioned-concurrency-config`. El siguiente comando asigna 100 unidades de simultaneidad aprovisionadas para el alias `BLUE` de una función llamada `my-function`:



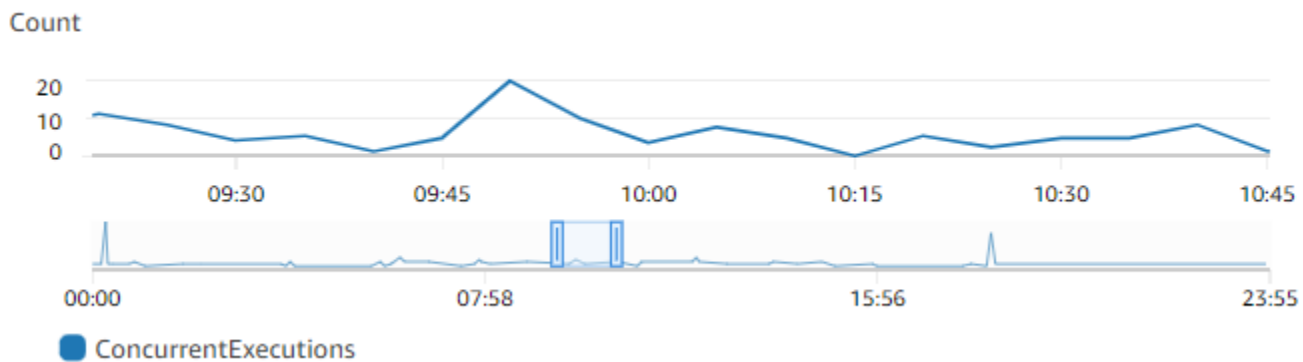
```
aws lambda put-provisioned-concurrency-config --function-name my-function \  
--qualifier BLUE \  
--provisioned-concurrent-executions 100
```

Debería ver una salida con un aspecto similar al siguiente:

```
{  
  "Requested ProvisionedConcurrentExecutions": 100,  
  "Allocated ProvisionedConcurrentExecutions": 0,  
  "Status": "IN_PROGRESS",  
  "LastModified": "2023-01-21T11:30:00+0000"  
}
```

## Estimar con precisión la simultaneidad aprovisionada requerida para una función

Puede ver las métricas de simultaneidad de cualquier función activa mediante las [métricas de CloudWatch](#). En concreto, la métrica `ConcurrentExecutions` muestra la cantidad de invocaciones simultáneas para las funciones de la cuenta.



Según el gráfico anterior, esta función atiende un promedio de 5 a 10 solicitudes simultáneas en cualquier momento y alcanza un máximo de 20 solicitudes. Supongamos que hay muchas otras funciones en la cuenta. Si esta función es esencial para la aplicación y necesita una respuesta de baja latencia en cada invocación, configure, al menos, 20 unidades de simultaneidad aprovisionada.

Recuerde que también puede [calcular la simultaneidad](#) mediante la siguiente fórmula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Para calcular cuánta simultaneidad necesita, multiplique el promedio de solicitudes por segundo por la duración promedio de las solicitudes en segundos. Puede estimar el promedio de solicitudes por segundo mediante la métrica de `Invocation` y la duración promedio de las solicitudes en segundos mediante la métrica de `Duration`.

Al configurar la simultaneidad aprovisionada, Lambda sugiere agregar un búfer del 10 % además de la cantidad de simultaneidad que normalmente necesita la función. Por ejemplo, si la función suele alcanzar un máximo de 200 solicitudes simultáneas, establezca la simultaneidad aprovisionada en 220 (200 solicitudes simultáneas + un 10 % = 220 de simultaneidad aprovisionada).

## Optimizar el código de la función cuando se utiliza la simultaneidad aprovisionada

Si utilizas la simultaneidad aprovisionada, considera la posibilidad de reestructurar el código de función para optimizarlo y lograr una latencia baja. Para las funciones que se ejecutan con simultaneidad aprovisionada, Lambda ejecuta cualquier código de inicialización (por ejemplo, cargar bibliotecas y crear instancias de clientes) en el momento de la asignación. Por lo tanto, se recomienda mover la mayor cantidad de inicialización fuera del controlador de funciones principal para evitar que afecte a la latencia durante las invocaciones reales de la función. Por el contrario, inicializar bibliotecas o crear instancias de clientes dentro del código de su controlador principal significa que la función debe ejecutarlo cada vez que la invoque, independientemente de si utiliza o no la simultaneidad aprovisionada.

Para las invocaciones bajo demanda, es posible que Lambda tenga que volver a ejecutar el código de inicialización cada vez que la función se inicie en frío. Para estas funciones, puede optar por aplazar la inicialización de una capacidad específica hasta que su función lo necesite. Por ejemplo, considere el siguiente flujo de control para un controlador de Lambda:

```
def handler(event, context):
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
    else:
        // Do nothing
```

En el ejemplo anterior, en lugar de inicializar `CLIENT_A` fuera del controlador principal, el desarrollador lo inicializó dentro de la instrucción `if`. De este modo, Lambda ejecuta este código solo si se cumple `some_condition`. Si inicializa `CLIENT_A` fuera del controlador principal, Lambda ejecuta ese código cada vez que se inicia en frío. Esto puede aumentar la latencia general.

## Usar variables de entorno para ver y controlar el comportamiento de simultaneidad aprovisionado

Es posible que la función utilice toda la simultaneidad aprovisionada. Lambda usa instancias bajo demanda para gestionar cualquier exceso de tráfico. Para determinar el tipo de inicialización que Lambda usó para un entorno determinado, compruebe el valor de la variable de entorno `AWS_LAMBDA_INITIALIZATION_TYPE`. Esta variable tiene dos valores posibles: `provisioned-concurrency` u `on-demand`. El valor de `AWS_LAMBDA_INITIALIZATION_TYPE` es inmutable y constante durante toda la vida útil del entorno. Para comprobar el valor de una variable de entorno en el código de su función, consulte [???](#).

Si está utilizando los tiempos de ejecución de .NET 6 o .NET 7, puede configurar la variable de entorno `AWS_LAMBDA_DOTNET_PREJIT` para mejorar la latencia de las funciones, incluso si no utilizan la simultaneidad aprovisionada. El tiempo de ejecución .NET utiliza compilación e inicialización perezosa para cada biblioteca a la que llama el código por primera vez. Como resultado, la primera invocación de una función de Lambda puede tardar más que las posteriores. Para mitigar este problema, puede elegir uno de los tres valores para `AWS_LAMBDA_DOTNET_PREJIT`:

- `ProvisionedConcurrency`: Lambda realiza una compilación JIT anticipada para todos los entornos mediante la simultaneidad aprovisionada. Este es el valor predeterminado.
- `Always`: Lambda realiza una compilación JIT anticipada para cada entorno, incluso si la función no usa la simultaneidad aprovisionada.
- `Never`: Lambda deshabilita la compilación JIT anticipada para todos los entornos.

## Comprenda el comportamiento de registro y facturación con la simultaneidad aprovisionada

Para los entornos de simultaneidad aprovisionada, el código de inicialización de su función se ejecuta durante la asignación y de manera periódica, a medida que las instancias de la función se reciclan. Puede ver el tiempo de inicialización en registros y [seguimientos](#) después de que una instancia de entorno procese una solicitud. Es importante tener en cuenta que Lambda factura la inicialización incluso si la instancia de entorno nunca procesa una solicitud. La simultaneidad aprovisionada se ejecuta continuamente y se factura por separado de los costos de inicialización e invocación. Para obtener más información, consulte [Precios de AWS Lambda](#).

Además, al configurar una función de Lambda con una simultaneidad aprovisionada, Lambda preinicializa ese entorno de ejecución para que esté disponible antes de las solicitudes de invocación de funciones. Sin embargo, la función publica los registros de invocación en CloudWatch solo cuando la función se invoca de verdad. Por lo tanto, el [campo Duración de la inicialización](#) aparece en la línea de registro REPORT de la primera invocación de la función, aunque la inicialización se haya realizado antes de tiempo. Esto no significa que la función haya tenido un inicio en frío.

## Uso de Application Auto Scaling para automatizar la administración de simultaneidad aprovisionada

El Auto Scaling de aplicaciones le permite administrar la simultaneidad aprovisionada según una programación o en función de la utilización. Si la función recibe patrones de tráfico predecibles, use el escalado programado. Si quiere que la función mantenga un porcentaje de utilización específico, utilice una política de escalado de seguimiento de destino.

### Escalado programado

Con el Auto Scaling de aplicaciones, puede definir su propia programación de escalado según los cambios predecibles en las cargas. Para obtener más información y ejemplos, consulte [Escalado programado para el Auto Scaling de aplicaciones](#) en la Guía del usuario de Auto Scaling de aplicaciones y [Programación de la simultaneidad aprovisionada de AWS Lambda para picos de uso recurrentes](#) en el blog de AWS Compute.

### Seguimiento de destino

Con el seguimiento de destino, el Auto Scaling de aplicaciones crea y administra un conjunto de alarmas de CloudWatch según la forma en que usted define la política de escalado. Cuando se activan estas alarmas, el Auto Scaling de aplicaciones ajusta automáticamente la cantidad de entornos asignados mediante la simultaneidad aprovisionada. Utilice el seguimiento de destino para aplicaciones que no tienen patrones de tráfico predecibles.

Para escalar la simultaneidad aprovisionada mediante el seguimiento de destino, use las operaciones `RegisterScalableTarget` y `PutScalingPolicy` de la API de Auto Scaling de aplicaciones. Por ejemplo, si usa la AWS Command Line Interface (CLI), siga los siguientes pasos:

1. Registre el alias de una función como destino del escalado. En el ejemplo siguiente se registra el alias BLUE de una función denominada `my-function`:

```
aws application-autoscaling register-scalable-target --service-namespace lambda \
```

```
--resource-id function:my-function:BLUE --min-capacity 1 --max-capacity 100 \
--scalable-dimension lambda:function:ProvisionedConcurrency
```

2. Aplique una política de escalado al destino. En el ejemplo siguiente, se configura el escalado automático de aplicaciones para ajustar la configuración de simultaneidad aprovisionada para que un alias mantenga la utilización cerca del 70 %, pero puede aplicar cualquier valor entre 10 % y 90 %.

```
aws application-autoscaling put-scaling-policy \
  --service-namespace lambda \
  --scalable-dimension lambda:function:ProvisionedConcurrency \
  --resource-id function:my-function:BLUE \
  --policy-name my-policy \
  --policy-type TargetTrackingScaling \
  --target-tracking-scaling-policy-configuration '{ "TargetValue":
0.7, "PredefinedMetricSpecification": { "PredefinedMetricType":
"LambdaProvisionedConcurrencyUtilization" } }'
```

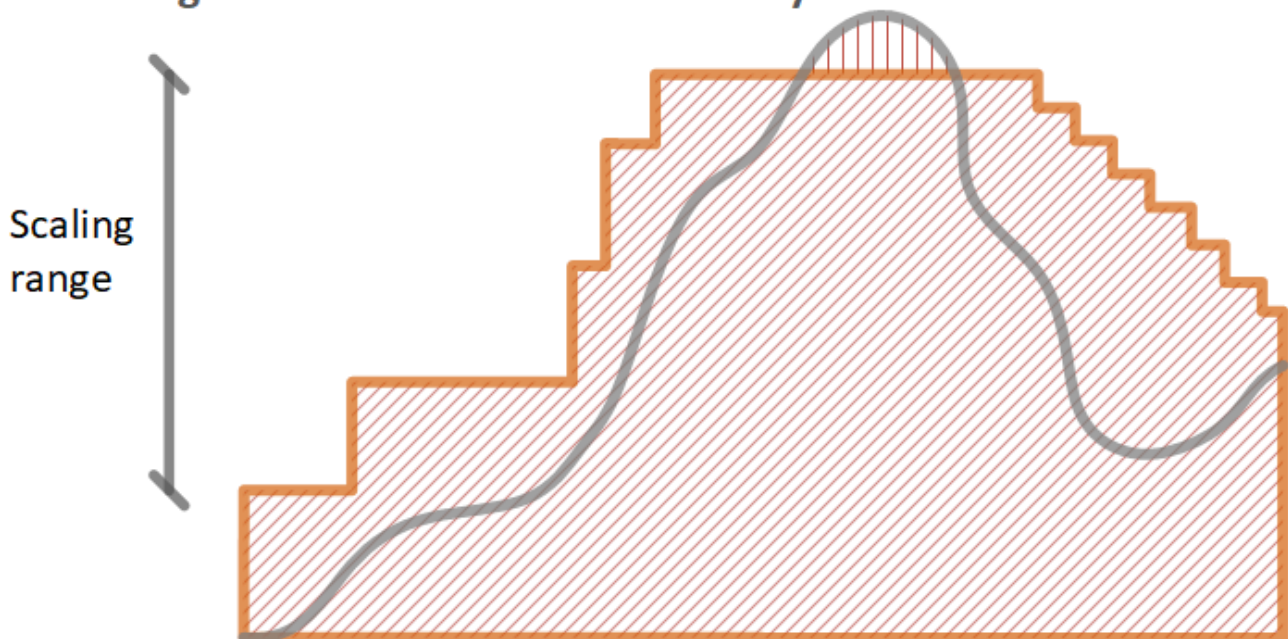
Debería ver un resultado con un aspecto similar al siguiente:

```
{
  "PolicyARN": "arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:12266dbb-1524-xmpl-a64e-9a0a34b996fa:resource/lambda/
function:my-function:BLUE:policyName/my-policy",
  "Alarms": [
    {
      "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
      "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7"
    },
    {
      "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66",
      "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66"
    }
  ]
}
```




Auto Scaling de aplicaciones crea dos alarmas en CloudWatch. La primera alarma se desencadena cuando la utilización de la simultaneidad aprovisionada supera sistemáticamente el 70 %. Cuando esto sucede, Auto Scaling de aplicaciones asigna más simultaneidad aprovisionada para reducir esta utilización. La segunda alarma se desencadena cuando la utilización nunca supera el 63 % (un 90 % del objetivo del 70 %). Cuando esto sucede, Auto Scaling de aplicaciones reduce la concurrencia aprovisionada del alias.

En el ejemplo siguiente, una función escala entre una cantidad mínima y máxima de concurrencia aprovisionada basada en la utilización.

### Autoscaling with Provisioned Concurrency



#### Leyenda

-  Instancias de función
-  Solicitudes abiertas
-  Simultaneidad aprovisionada

- |||||

### Simultaneidad estándar

Cuando aumenta el número de solicitudes abiertas, el Auto Scaling de aplicaciones aumenta la simultaneidad aprovisionada en grandes pasos hasta que alcanza el máximo configurado. Luego, la función podrá seguir escalando según la simultaneidad estándar no reservada si la cuenta no alcanzó su límite de simultaneidad. Cuando la utilización disminuye y permanece baja, el escalado automático de aplicaciones disminuye la simultaneidad aprovisionada en pasos periódicos más pequeños.

Las dos alarmas del escalado automático de aplicaciones utilizan la estadística promedio de forma predeterminada. Es posible que las funciones que experimentan ráfagas no desencadenen estas alarmas. Por ejemplo, supongamos que la función de Lambda se ejecuta rápidamente (es decir, de 20 a 100 ms) y que el tráfico se produce en ráfagas rápidas. En este caso, el número de solicitudes supera la simultaneidad aprovisionada asignada durante la ráfaga. Sin embargo, el escalado automático de aplicaciones requiere que la carga de ráfaga se mantenga durante al menos 3 minutos para aprovisionar entornos adicionales. Además, ambas alarmas de CloudWatch requieren 3 puntos de datos que alcancen el promedio de destino para activar la política de escalado automático. Si su función experimenta ráfagas rápidas de tráfico, usar la estadística Máximo en lugar de la estadística Promedio puede resultar más eficaz a la hora de escalar la simultaneidad aprovisionada y minimizar los arranques en frío.

Para obtener más información acerca de las políticas de escalado de seguimiento de destino, consulte [Políticas de escalado de seguimiento de destino para el Auto Scaling de aplicaciones](#).

# Comportamiento de escalado de Lambda

A medida que una función recibe más solicitudes, Lambda escala de forma automática la cantidad de entornos de ejecución para gestionar estas solicitudes hasta que su cuenta alcance la cuota de simultaneidad. Sin embargo, para protegerse contra el exceso de escalado en respuesta a ráfagas repentinas de tráfico, Lambda limita la rapidez con la que sus funciones pueden escalar. Esta tasa de escalado de simultaneidad es la tasa máxima a la que las funciones de su cuenta pueden escalar en respuesta a un aumento de las solicitudes. (Es decir, la rapidez con la que Lambda puede crear nuevos entornos de ejecución). La tasa de escalado de simultaneidad difiere del límite de simultaneidad de la cuenta, que es la cantidad total de simultaneidad disponible para sus funciones.

## Tasa de escalado de simultaneidad

En cada Región de AWS y para cada función, su tasa de escalado de simultaneidad es de 1000 instancias del entorno de ejecución cada 10 segundos (o 10 000 solicitudes por segundo cada 10 segundos). En otras palabras, cada 10 segundos, Lambda puede asignar como máximo 1000 instancias de entorno de ejecución adicionales o admitir 10 000 solicitudes adicionales a cada una de sus funciones.

Por lo general, no debe preocuparse por esta limitación. La tasa de escalado de Lambda es suficiente para la mayoría de los casos de uso.

Es importante destacar que la tasa de escalado de simultaneidad es un límite de función. Esto significa que cada función de su cuenta puede escalar independientemente de otras funciones.

### Note

En la práctica, Lambda hace todo lo posible por reponer la tasa de escalado de simultaneidad de forma continua a lo largo del tiempo, en lugar de hacer una sola reposición de 1000 unidades cada 10 segundos.

Lambda no acumula partes no utilizadas de la tasa de escalado de simultaneidad. Esto significa que, en cualquier momento, su tasa de escalado es siempre de 1000 unidades de simultaneidad como máximo. Por ejemplo, si no utiliza ninguna de las 1000 unidades de simultaneidad disponibles en un intervalo de 10 segundos, no acumulará 1000 unidades adicionales en el siguiente intervalo de 10 segundos. Su tasa de escalado de simultaneidad seguirá siendo de 1000 en el siguiente intervalo de 10 segundos.



Mientras su función continúe recibiendo un número cada vez mayor de solicitudes, Lambda escalará a la tasa más rápida disponible hasta el límite de simultaneidad de su cuenta. Puede limitar la cantidad de simultaneidad que pueden utilizar las funciones individuales [configurando la simultaneidad reservada](#). Si llegan solicitudes más rápidamente de lo que la función puede escalar o si la función está en la simultaneidad máxima, entonces las solicitudes adicionales fallan con un error de limitación (código de estado 429).

# Monitoreo de la simultaneidad

Lambda emite métricas de Amazon CloudWatch para ayudarlo a monitorear la simultaneidad de sus funciones. En este tema, se explican estas métricas y cómo interpretarlas.

## Secciones

- [Métricas generales de simultaneidad](#)
- [Métricas de simultaneidad aprovisionada](#)
- [Cómo trabajar con la métrica ClaimedAccountConcurrency](#)

## Métricas generales de simultaneidad

Utilice estas métricas para monitorear la simultaneidad de las funciones de Lambda. La granularidad de cada métrica es de 1 minuto.

- `ConcurrentExecutions`: la cantidad de invocaciones simultáneas activas en un momento determinado. Lambda emite esta métrica para todas las funciones, alias y versiones. Para cualquier función de la consola de Lambda, se muestra el gráfico sobre `ConcurrentExecutions` de forma nativa en la pestaña Monitoreo, en Métricas. Visualice esta métrica mediante MAX.
- `UnreservedConcurrentExecutions`: la cantidad de invocaciones simultáneas activas que utilizan la simultaneidad no reservada. Lambda emite esta métrica en todas las funciones de una región. Visualice esta métrica mediante MAX.
- `ClaimedAccountConcurrency`: la cantidad de simultaneidad que no está disponible para las invocaciones bajo demanda. `ClaimedAccountConcurrency` es igual a `UnreservedConcurrentExecutions` más la cantidad de simultaneidad asignada (es decir, la simultaneidad reservada total más la simultaneidad aprovisionada total). Si `ClaimedAccountConcurrency` supera el límite de simultaneidad de su cuenta, puede [solicitar un límite de simultaneidad de cuentas superior](#). Visualice esta métrica mediante MAX. Para obtener más información, consulte [Cómo trabajar con la métrica ClaimedAccountConcurrency](#).

## Métricas de simultaneidad aprovisionada

Utilice las siguientes métricas para monitorear las funciones de Lambda mediante la simultaneidad aprovisionada. La granularidad de cada métrica es de 1 minuto.

- **ProvisionedConcurrentExecutions**: la cantidad de instancias de entorno de ejecución que están procesando de forma activa una invocación con simultaneidad aprovisionada. Lambda emite esta métrica para cada versión y alias de función con la simultaneidad aprovisionada configurada. Visualice esta métrica mediante MAX.

**ProvisionedConcurrentExecutions** no es lo mismo que el número total de simultaneidad aprovisionada que usted asigna. Por ejemplo, supongamos que asigna 100 unidades de simultaneidad aprovisionadas a una versión de la función. Durante un minuto dado, si al menos 50 de esos 100 entornos de ejecución gestionaban invocaciones en simultáneo, el valor de MAX (**ProvisionedConcurrentExecutions**) es 50.

- **ProvisionedConcurrentInvocations**: es la cantidad de veces que Lambda invoca el código de la función mediante la simultaneidad aprovisionada. Lambda emite esta métrica para cada versión y alias de función con la simultaneidad aprovisionada configurada. Visualice esta métrica mediante SUM.

**ProvisionedConcurrentInvocations** difiere de **ProvisionedConcurrentExecutions** en que **ProvisionedConcurrentInvocations** cuenta el número total de invocaciones, mientras que **ProvisionedConcurrentExecutions** cuenta el número de entornos activos. Para entender esta distinción, considere el siguiente escenario:



En este ejemplo, suponga que recibe 1 invocación por minuto y que cada invocación tarda 2 minutos en completarse. Cada barra horizontal naranja representa una única solicitud. Supongamos que asigna 10 unidades de simultaneidad aprovisionadas a esta función, de modo que cada solicitud se ejecute con simultaneidad aprovisionada.

Entre los minutos 0 y 1, entra Request 1. En el minuto 1, el valor de MAX (ProvisionedConcurrentExecutions) es 1, ya que, como máximo, 1 entorno de ejecución estuvo activo durante el último minuto. El valor de SUM (ProvisionedConcurrentInvocations) también es 1, ya que se recibió 1 solicitud nueva en el último minuto.

Entre los minutos 1 y 2, entra la Request 2 y la Request 1 continúa ejecutándose. En el minuto 2, el valor de MAX (ProvisionedConcurrentExecutions) es 2, ya que durante el último minuto estuvieron activos como máximo 2 entornos de ejecución. Sin embargo, el valor de SUM (ProvisionedConcurrentInvocations) es 1, ya que solo se recibió 1 solicitud nueva en el último minuto. Este comportamiento de la métrica continúa hasta el final del ejemplo.

- **ProvisionedConcurrencySpilloverInvocations**: es la cantidad de veces que Lambda invoca la función en simultaneidad estándar (reservada o no reservada) cuando toda la simultaneidad aprovisionada está en uso. Lambda emite esta métrica para cada versión y alias de función con la simultaneidad aprovisionada configurada. Visualice esta métrica mediante SUM. El valor de ProvisionedConcurrentInvocations + ProvisionedConcurrencySpilloverInvocations debe ser igual al número total de invocaciones de funciones (es decir, la métrica Invocations).

**ProvisionedConcurrencyUtilization**: el porcentaje de simultaneidad aprovisionada en uso (por ejemplo, el valor de ProvisionedConcurrentExecutions dividido por la cantidad total de simultaneidad aprovisionada asignada). Lambda emite esta métrica para cada versión y alias de función con la simultaneidad aprovisionada configurada. Visualice esta métrica mediante MAX.

Por ejemplo, supongamos que aprovisiona 100 unidades de simultaneidad aprovisionadas a una versión de la función. Durante un minuto dado, si como máximo 60 de esos 100 entornos de ejecución gestionaban invocaciones de forma simultánea, el valor de MAX (ProvisionedConcurrentExecutions) es 60 y el valor de MAX (ProvisionedConcurrentUtilization) es 0,6.

Un valor alto para ProvisionedConcurrencySpilloverInvocations puede indicar que necesita asignar una simultaneidad aprovisionada adicional para su función. Como alternativa, puede [configurar el Auto Scaling de aplicaciones para gestionar el escalado automático de la simultaneidad aprovisionada](#) en función de umbrales predefinidos.

Por el contrario, los valores consistentemente bajos de ProvisionedConcurrencyUtilization pueden indicar que ha asignado en exceso la simultaneidad aprovisionada para su función.

## Cómo trabajar con la métrica **ClaimedAccountConcurrency**

Lambda utiliza la métrica `ClaimedAccountConcurrency` para determinar la cantidad de simultaneidad disponible de su cuenta para las invocaciones bajo demanda. Lambda calcula `ClaimedAccountConcurrency` mediante la siguiente fórmula:

$$\text{ClaimedAccountConcurrency} = \text{UnreservedConcurrentExecutions} + (\text{allocated concurrency})$$

`UnreservedConcurrentExecutions` es la cantidad de invocaciones simultáneas activas que utilizan la simultaneidad no reservada. La simultaneidad asignada es la suma de las siguientes dos partes (RC se sustituye por “simultaneidad reservada” y PC por “simultaneidad aprovisionada”):

- El RC total de todas las funciones de una Región.
- El PC total de todas las funciones de una Región que utilizan PC, excepto las funciones que utilizan RC.

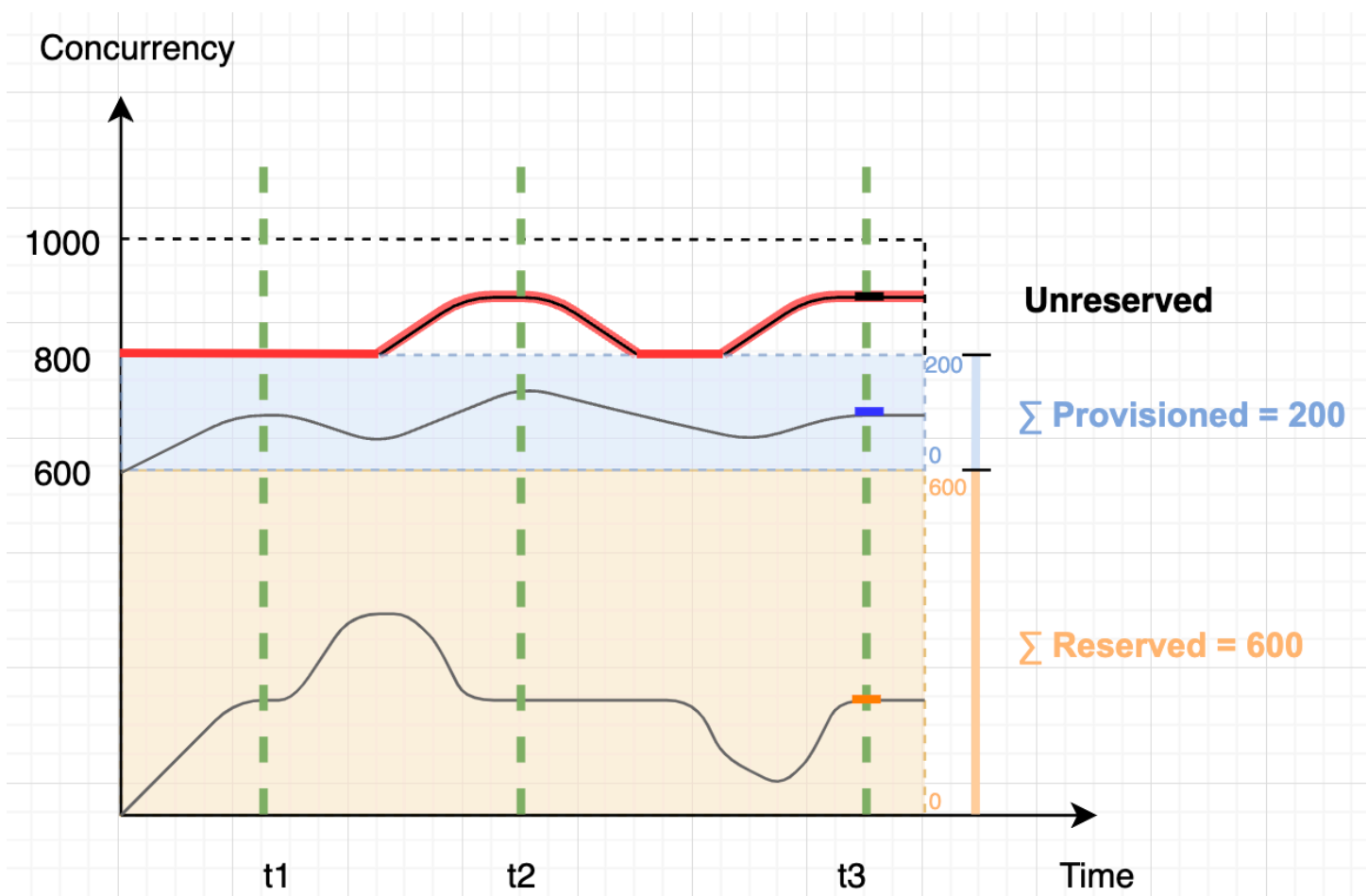
### Note

No puede asignar más PC que RC para una función. Por lo tanto, la RC de una función siempre es mayor o igual que su PC. Para calcular la contribución a la simultaneidad asignada para dichas funciones con PC y RC, Lambda solo tiene en cuenta RC, que es la más alta de las dos.

Lambda utiliza la métrica `ClaimedAccountConcurrency`, en lugar de `ConcurrentExecutions`, para determinar la cantidad de simultaneidad disponible para las invocaciones bajo demanda. Si bien la métrica `ConcurrentExecutions` es útil para realizar un seguimiento del número de invocaciones simultáneas activas, no siempre refleja su verdadera disponibilidad simultánea. Esto se debe a que Lambda también considera la simultaneidad reservada y la simultaneidad aprovisionada para determinar la disponibilidad.

Para ejemplificar `ClaimedAccountConcurrency`, imagine un escenario en el que configura una gran cantidad de simultaneidad reservada y aprovisionada en todas las funciones que prácticamente no se utilizan. En el siguiente ejemplo, supongamos que el límite de simultaneidad de su cuenta es 1000 y que tiene dos funciones principales en su cuenta: `function-orange` y `function-blue`. Asigna 600 unidades de simultaneidad reservada a `function-orange`. Asigna 200 unidades de

simultaneidad provisionada a `function-blue`. Supongamos que, con el transcurso del tiempo, implementa funciones adicionales y observa el siguiente patrón de tráfico:



En el diagrama anterior, las líneas negras indican el uso real de la simultaneidad a lo largo del tiempo y la línea roja indica el valor de `ClaimedAccountConcurrency` a lo largo del tiempo. En este escenario, `ClaimedAccountConcurrency` es 800 como mínimo, a pesar del bajo uso real de la simultaneidad en todas las funciones. Esto se debe a que asignó 800 unidades totales de simultaneidad para `function-orange` y `function-blue`. Desde el punto de vista de Lambda, usted “reclamó” esta simultaneidad para utilizarla, por lo que, efectivamente, solo le quedan 200 unidades de simultaneidad para otras funciones.

En este escenario, la simultaneidad asignada es 800 en la fórmula `ClaimedAccountConcurrency`. A continuación, podemos derivar el valor de `ClaimedAccountConcurrency` en varios puntos del diagrama:

- En  $t_1$ , `ClaimedAccountConcurrency` es 800 ( $800 + 0$  `UnreservedConcurrentExecutions`).

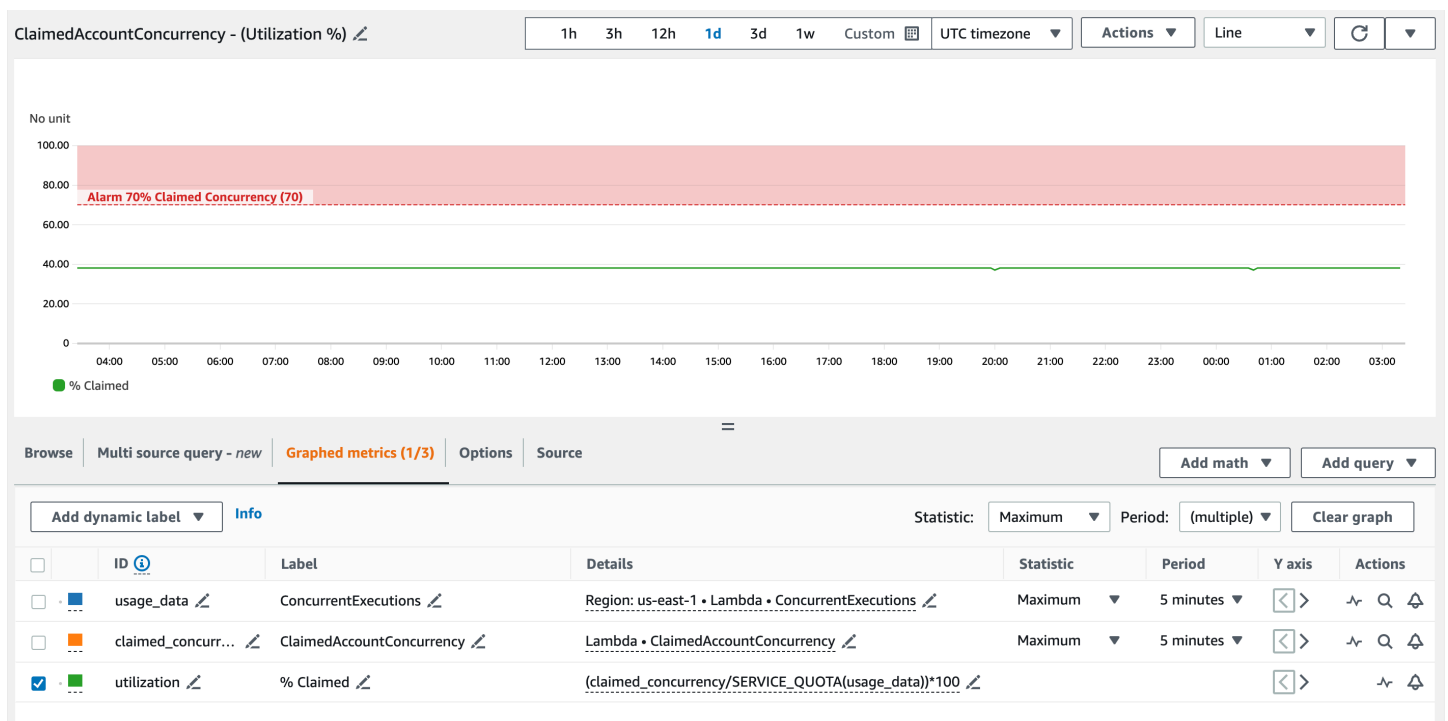
- En t2, `ClaimedAccountConcurrency` es 900 ( $800 + 100$  `UnreservedConcurrentExecutions`).
- En t3, `ClaimedAccountConcurrency` es nuevamente 900 ( $800 + 100$  `UnreservedConcurrentExecutions`).

## Configuración de la métrica `ClaimedAccountConcurrency` en CloudWatch

Lambda emite la métrica `ClaimedAccountConcurrency` en CloudWatch. Utilice esta métrica junto con el valor de `SERVICE_QUOTA(ConcurrentExecutions)` para obtener el porcentaje de utilización de la simultaneidad en su cuenta, como se muestra en la siguiente fórmula:

$$\text{Utilization} = (\text{ClaimedAccountConcurrency} / \text{SERVICE\_QUOTA}(\text{ConcurrentExecutions})) * 100\%$$

La siguiente captura de pantalla muestra cómo se puede graficar esta fórmula en CloudWatch. La línea verde `claim_utilization` representa la utilización simultánea de esta cuenta, que es alrededor de 40 %:



La captura de pantalla anterior también incluye una alarma de CloudWatch que pasa a estado ALARM cuando la utilización de la simultaneidad supera el 70 %. Puede utilizar la métrica `ClaimedAccountConcurrency` junto con alarmas similares para determinar de forma proactiva cuándo podría ser necesario solicitar un límite de simultaneidad de cuentas superior.

# Creación de funciones de Lambda con Node.js

Puede ejecutar código JavaScript con Node.js en AWS Lambda. Lambda proporciona [tiempos de ejecución](#) para Node.js que ejecutan su código para procesar eventos. El código se ejecuta en un entorno que incluye AWS SDK for JavaScript, con credenciales de un rol de AWS Identity and Access Management (IAM) que usted administre. Para obtener más información sobre las versiones del SDK incluidas en los tiempos de ejecución de Node.js, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Lambda admite los siguientes tiempos de ejecución de Node.js.

Node.js

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Node.js 20	nodejs20.x	Amazon Linux 2023			
Node.js 18	nodejs18.x	Amazon Linux 2			

## Note

Los tiempos de ejecución de Node.js 18 y versiones posteriores usan el AWS SDK para JavaScript v3. Para migrar una función desde un tiempo de ejecución anterior, siga el [taller de migración](#) en GitHub. Para obtener más información sobre la versión 3 de AWS SDK para JavaScript, consulte la publicación de blog [La versión modular de AWS SDK para JavaScript ya está disponible para el público general](#).

Para crear una función de Node.js.

1. Abra la [consola de Lambda](#).
2. Elija Crear función.
3. Configure los siguientes ajustes:



- En Nombre de la función: ingrese el nombre de la función.
  - Tiempo de ejecución: elija Node.js 20.x.
4. Elija Crear función.
  5. Para configurar un evento de prueba, seleccione Prueba.
  6. En Nombre del evento, escriba **test**.
  7. Elija Guardar cambios.
  8. Elija Test (Probar) para invocar la función.

La consola crea una función de Lambda con un único archivo de origen llamado `index.js` o `index.mjs`. Puede editar este archivo y agregar más archivos en el editor de código integrado. Para guardar los cambios, elija Guardar. A continuación, para ejecutar el código, elija Pruebas.

#### Note

La consola de Lambda utiliza AWS Cloud9 para proporcionar un entorno de desarrollo integrado en el navegador. También puede utilizar AWS Cloud9 para desarrollar funciones de Lambda en su propio entorno. Para obtener más información, consulte [Trabajar con funciones de AWS Lambda utilizando AWS Toolkit](#) en la guía del usuario de AWS Cloud9.

El archivo `index.js` o `index.mjs` exporta una función denominada `handler` que toma un objeto `event` y un objeto `context`. Esta es la [función de controlador](#) a la que llama Lambda cuando se invoca la función. El tiempo de ejecución de la función de Node.js obtiene los eventos de invocación de Lambda y se los pasa al controlador. En la configuración de función, el valor de controlador es `index.handler`.

Al guardar el código de función, la consola de Lambda crea un paquete de implementación de archivo `.zip`. Cuando desarrolle el código de función fuera de la consola (mediante un IDE), debe [crear un paquete de implementación](#) para cargar el código a la función de Lambda.

#### Note

Para comenzar con el desarrollo de aplicaciones en su entorno local, implemente una de las aplicaciones de ejemplo disponibles en el repositorio de GitHub de esta guía.

## Aplicaciones de Lambda de ejemplo en Node.js

- [blank-nodejs](#): una función de Node.js que muestra el uso de registro, las variables de entorno, el seguimiento de AWS X-Ray, las capas, las pruebas de unidad y el AWS SDK.
- [nodejs-apig](#): una función con un punto de conexión de API público que procesa un evento desde API Gateway y devuelve una respuesta HTTP.
- [efs-nodejs](#): una función que utiliza un sistema de archivos de Amazon EFS en una Amazon VPC. Este ejemplo incluye una VPC, un sistema de archivos, objetivos de montaje y un punto de acceso configurados para su uso con Lambda.

El tiempo de ejecución de la función pasa un objeto context al controlador, además del evento de invocación. El [objeto context](#) contiene información adicional acerca de la invocación, la función y el entorno de ejecución. Hay más información disponible en las variables de entorno.

Su función de Lambda tiene un grupo de registros de Registros de CloudWatch. El tiempo de ejecución de la función envía detalles de cada invocación a Registros de CloudWatch. Se transmite cualquier [registro que su función genere](#) durante la invocación. Si su función devuelve un error, Lambda formatea el error y lo devuelve al invocador.

## Temas

- [Inicialización de Node.js](#)
- [Versiones del SDK incluidas en el tiempo de ejecución](#)
- [Uso de keep-alive para conexiones TCP](#)
- [Carga del certificado de la CA](#)
- [Definir el controlador de las funciones de Lambda en Node.js](#)
- [Implementar funciones Node.js de Lambda con archivos de archivo.zip](#)
- [Implementar funciones Node.js Lambda con imágenes de contenedor](#)
- [Uso del objeto de contexto Lambda para recuperar la información de la función Node.js](#)
- [Registro y supervisión de las funciones de Lambda de Node.js](#)
- [Instrumentación del código Node.js en AWS Lambda](#)

# Inicialización de Node.js

Node.js tiene un modelo de bucle de eventos único que hace que su comportamiento de inicialización sea diferente al de otros tiempos de ejecución. En concreto, Node.js utiliza un modelo de E/S sin bloqueo que admite operaciones asíncronas. Este modelo permite que Node.js funcione de forma eficiente para la mayoría de las cargas de trabajo. Por ejemplo, si una función de Node.js lleva a cabo una llamada de red, esa solicitud puede designarse como operación asíncrona y colocarse en una cola de devolución de llamadas. Es posible que la función continúe procesando otras operaciones dentro de la pila de llamadas principal sin bloquearse mientras espera a que se devuelva la llamada de red. Una vez que se completa la llamada de red, se ejecuta la devolución de llamada y, a continuación, se elimina de la cola de devolución de llamadas.

Algunas tareas de inicialización pueden ejecutarse de forma asíncrona. No se garantiza que estas tareas asíncronas completen la ejecución antes de una invocación. Por ejemplo, es posible que el código que lleva a cabo la llamada de red para recuperar un parámetro de AWS Parameter Store no esté completo en el momento en el que Lambda ejecuta la función de controlador. Como resultado, la variable puede ser nula durante una invocación. Para evitarlo, asegúrese de que las variables y otros códigos asíncronos se inicialicen por completo antes de continuar con el resto de la lógica empresarial principal de la función.

Como alternativa, puede designar el código de la función como módulo ES, lo que le permite usar `await` en el nivel superior del archivo, fuera del ámbito del controlador de la función. Cuando `await` todas las `Promise`, el código de inicialización asíncrono se completa antes de las invocaciones del controlador, lo que maximiza la eficacia de la [simultaneidad aprovisionada](#) para reducir la latencia de inicio en frío. Para obtener más información y un ejemplo, consulte [Uso de módulos ES de Node.js y espera de nivel superior en AWS Lambda](#).

## Designación de un controlador de funciones como módulo de ES

Lambda trata los archivos con el sufijo `.js` como módulos CommonJS de forma predeterminada. Si así lo desea, puede designar su código como módulo ES. Puede hacerlo de dos formas: especificar el `type` como `module` en el archivo `package.json` de la función o usar la extensión de nombre de archivo `.mjs`. En el primer enfoque, el código de la función trata todos los archivos `.js` como módulos ES, mientras que, en el segundo caso, solo el archivo que especifique con `.mjs` es un módulo ES. Para mezclar módulos ES y CommonJS, asígneles los nombres `.mjs` y `.cjs` respectivamente, ya que los archivos `.mjs` son siempre módulos ES y los archivos `.cjs` son siempre módulos CommonJS.

Lambda busca carpetas en la variable de entorno `NODE_PATH` al cargar los módulos ES. Puede cargar el SDK de AWS que se incluye en el tiempo de ejecución mediante las instrucciones de `import` del módulo ES. También puede cargar módulos ES a partir de [capas](#).

## Versiones del SDK incluidas en el tiempo de ejecución

La versión del AWS SDK incluida en el tiempo de ejecución de Node.js depende de la versión del tiempo de ejecución y de su Región de AWS. Para encontrar la versión del SDK incluida en el tiempo de ejecución que está utilizando, cree una función de Lambda con el código siguiente.

### Note

El código de ejemplo que se muestra a continuación para las versiones 18 y superiores de Node.js usa el formato CommonJS. Si crea la función en la consola de Lambda, asegúrese de cambiar el nombre del archivo que contiene el código a `index.js`.

### Example Node.js 18 y versiones superiores

```
const { version } = require("@aws-sdk/client-s3/package.json");

exports.handler = async () => ({ version });
```

Esta acción devuelve una respuesta con el siguiente formato:

```
{
  "version": "3.462.0"
}
```

## Uso de keep-alive para conexiones TCP

El agente HTTP o HTTPS predeterminado de Node.js crea una nueva conexión TCP para cada nueva solicitud. Para evitar el costo de establecer nuevas conexiones, puede utilizar `keepAlive: true` para reutilizar las conexiones que realiza su función mediante AWS SDK para JavaScript. Keep-alive puede reducir los tiempos de solicitud de las funciones de Lambda que realizan varias llamadas a la API mediante el SDK.

En el AWS SDK para JavaScript 3.x, que se incluye en `nodejs18.x` y tiempos de ejecución de Lambda posteriores, keep-alive está habilitado de forma predeterminada. Para deshabilitar keep-

alive, consulte [Reusing connections with keep-alive in Node.js](#) en la Guía para desarrolladores de AWS SDK para JavaScript 3.x. Para obtener más información sobre el uso de keep-alive, consulte [HTTP keep-alive is on by default in modular AWS SDK for JavaScript](#) en AWS Developer Tools Blog.

## Carga del certificado de la CA

Para las versiones de tiempo de ejecución hasta Node.js 18, Lambda carga automáticamente los certificados de la CA (autoridad de certificación) específicos de Amazon para facilitar la creación de funciones que interactúen con otros servicios de AWS. Por ejemplo, Lambda incluye los certificados de Amazon RDS necesarios para validar el [certificado de identidad del servidor](#) instalado en la base de datos de Amazon RDS. Este comportamiento puede afectar el rendimiento durante los arranques en frío.

A partir de Node.js 20, Lambda ya no carga certificados de la CA adicionales de forma predeterminada. El tiempo de ejecución Node.js 20 contiene un archivo de certificado con todos los certificados de la CA de Amazon ubicados en `/var/runtime/ca-cert.pem`. Para restaurar el mismo comportamiento de los tiempos de ejecución de Node.js 18 y anteriores, establezca la [variable `NODE\_EXTRA\_CA\_CERTS` del entorno](#) en `/var/runtime/ca-cert.pem`.

Para obtener un rendimiento óptimo, recomendamos que agrupe solo los certificados que necesite con su paquete de implementación y que los cargue mediante la variable de entorno `NODE_EXTRA_CA_CERTS`. El archivo de certificados debe constar de uno o más certificados de CA raíz o intermedios de confianza en formato PEM. Por ejemplo, en el caso de RDS, incluya los certificados necesarios junto con el código como `certificates/rds.pem`. A continuación, cargue los certificados configurando `NODE_EXTRA_CA_CERTS` como `/var/task/certificates/rds.pem`.

# Definir el controlador de las funciones de Lambda en Node.js

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

## Temas

- [Conceptos básicos del controlador de Node.js](#)
- [Denominación](#)
- [Uso de async/await](#)
- [Uso de devolución de llamadas](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Node.js](#)

## Conceptos básicos del controlador de Node.js

La siguiente función de ejemplo registra el contenido del [objeto de evento](#) y devuelve la ubicación de los registros.

### Note

Esta página muestra ejemplos de controladores de módulos CommonJS y ES. Para obtener más información sobre la diferencia entre estos dos tipos de controladores, consulte [Designación de un controlador de funciones como módulo de ES](#).

## ES module handler

### Example

```
export const handler = async (event, context) => {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

## CommonJS module handler

### Example

```
exports.handler = async function (event, context) {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

Al configurar una función, el valor del controlador es el nombre del archivo y el nombre del método del controlador exportado, separados por un punto. El valor predeterminado en la consola y para ejemplos de esta guía es `index.handler`. Esto indica el método `handler` que se exporta desde el archivo `index.js`.

El tiempo de ejecución pasa argumentos al método del controlador. El primer argumento es el objeto `event`, que contiene información del invoker. El invocador pasa esta información como una cadena con formato JSON cuando llama a [Invoke](#) y el runtime la convierte en un objeto. Cuando un servicio de AWS invoca su función, la estructura del evento [varía en función del servicio](#).

El segundo argumento es el [objeto context](#), que incluye información sobre la invocación, la función y el entorno de ejecución. En el ejemplo anterior, la función obtiene el nombre de [flujo de registro](#) del objeto `context` y lo devuelve al invocador.

También puede utilizar un argumento de devolución de llamada, una función a la que puede llamar en los controladores non-async para enviar una respuesta. Le recomendamos que utilice `async/await` en lugar de devolución de llamadas. `Async/await` mejora la legibilidad, el manejo de errores y la eficiencia. Para obtener información sobre las diferencias entre `async/await` y las devoluciones de llamadas, consulte [Uso de devolución de llamadas](#).

## Denominación

Al configurar una función, el valor del controlador es el nombre del archivo y el nombre del método del controlador exportado, separados por un punto. El valor predeterminado de las funciones creadas en la consola y de los ejemplos de esta guía es `index.handler`. Esto indica el método `handler` que se exporta desde el archivo `index.js` o `index.mjs`.

Si crea una función en la consola con un nombre de archivo o un nombre de controlador de funciones diferente, debe editar el nombre del controlador predeterminado.

## Para cambiar el nombre de controlador de la función (consola)

1. Abra la página [Funciones](#) de la consola de Lambda y elija su función.
2. Elija la pestaña Código.
3. Desplácese hacia abajo hasta el panel Configuración del tiempo de ejecución y elija Editar.
4. En Controlador, ingrese el nuevo nombre del controlador de funciones.
5. Seleccione Guardar.

## Uso de async/await

Si su código realiza una tarea asíncrona, utilice el patrón `async/await` para asegurarse de que el controlador termina de ejecutarse. `Async/await` es una forma concisa y legible de escribir código asíncrono en Node.js, sin necesidad de devoluciones de llamadas anidadas ni promesas encadenadas. Con `async/await`, puedes escribir código que se lea como código sincrónico, sin dejar de ser asíncrono y sin bloqueo.

La palabra clave `async` marca una función como asíncrona y la palabra clave `await` detiene la ejecución de la función hasta que se resuelva una `Promise`.

### Note

Asegúrese de esperar a que se completen los eventos asíncronos. Si la función regresa antes de que se completen los eventos asíncronos, podría fallar o causar un comportamiento inesperado en la aplicación. Esto puede suceder cuando un bucle `forEach` contiene un evento asíncrono. Los bucles `forEach` esperan una llamada sincrónica. Para obtener más información, consulte [Array.prototype.forEach\(\)](#) en la documentación de Mozilla.

## ES module handler

### Example — solicitud HTTP con `async/await`

```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
  try {
    // fetch is available in Node.js 18 and later runtimes
```



```
    const res = await fetch(url);
    console.info("status", res.status);
    return res.status;
  }
  catch (e) {
    console.error(e);
    return 500;
  }
};
```

## CommonJS module handler

### Example — solicitud HTTP con async/await

```
const https = require("https");
let url = "https://aws.amazon.com/";

exports.handler = async function (event) {
  let statusCode;
  await new Promise(function (resolve, reject) {
    https.get(url, (res) => {
      statusCode = res.statusCode;
      resolve(statusCode);
    }).on("error", (e) => {
      reject(Error(e));
    });
  });
  console.log(statusCode);
  return statusCode;
};
```

En el siguiente ejemplo, se usa `async/await` para enumerar los buckets de Amazon Simple Storage Service.

#### Note

Antes de usar este ejemplo, asegúrese de que el rol de ejecución de la función tenga permisos de lectura de Amazon S3.

## ES module handler

### Example — SDK de AWS v3 con async/await

En este ejemplo, se utiliza [AWS SDK for JavaScript v3](#), que está disponible en el tiempo de ejecución de `nodejs18.x` y posteriores.

```
import {S3Client, ListBucketsCommand} from '@aws-sdk/client-s3';
const s3 = new S3Client({region: 'us-east-1'});

export const handler = async(event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;
};
```

## CommonJS module handler

### Example — SDK de AWS v3 con async/await

En este ejemplo, se utiliza [AWS SDK for JavaScript v3](#), que está disponible en el tiempo de ejecución de `nodejs18.x` y posteriores.

```
const { S3Client, ListBucketsCommand } = require('@aws-sdk/client-s3');
const s3 = new S3Client({ region: 'us-east-1' });

exports.handler = async (event) => {
  const data = await s3.send(new ListBucketsCommand({}));
  return data.Buckets;
};
```

## Uso de devolución de llamadas

Le recomendamos que utilice [async/await](#) para declarar el controlador de funciones en lugar de utilizar devolución de llamadas. `Async/await` es una mejor opción por diversas razones:

- **Legibilidad:** el código `async/await` es más fácil de leer y entender que el código de devolución de llamada, que puede volverse difícil de seguir con rapidez y provocar un caos de devolución de llamadas.

- **Depuración y gestión de errores:** depurar código basado en devoluciones de llamadas puede resultar difícil. La pila de llamadas puede resultar difícil de seguir y los errores se pueden pasar por alto con facilidad. Con `async/await`, puede utilizar bloques `try/catch` para gestionar los errores.
- **Eficiencia:** las devoluciones de llamadas a menudo requieren cambiar entre diferentes partes del código. `Async/await` puede reducir la cantidad de cambios de contexto, lo que resulta en un código más eficiente.

Cuando utiliza devoluciones de llamadas en su controlador, la función prosigue con su ejecución hasta que el [bucle de eventos](#) está vacío o se agota el tiempo de espera de la función. La respuesta no se envía al invoker hasta que todas las tareas de bucle de eventos hayan terminado. Si el tiempo de espera de la función se agota, se devolverá un error en su lugar. Puede configurar el tiempo de ejecución para enviar la respuesta inmediatamente estableciendo [`context.callbackWaitsForEmptyEventLoop`](#) en `false`.

La función de devolución de llamada toma dos argumentos: un `Error` y una respuesta. El objeto de respuesta debe ser compatible con `JSON.stringify`.

La siguiente función de ejemplo comprueba una URL y devuelve el código de estado al invocador.

#### ES module handler

##### Example — solicitud HTTP con callback

```
import https from "https";
let url = "https://aws.amazon.com/";

export function handler(event, context, callback) {
  https.get(url, (res) => {
    callback(null, res.statusCode);
  }).on("error", (e) => {
    callback(Error(e));
  });
}
```

#### CommonJS module handler

##### Example — solicitud HTTP con callback

```
const https = require("https");
```

```
let url = "https://aws.amazon.com/";

exports.handler = function (event, context, callback) {
  https.get(url, (res) => {
    callback(null, res.statusCode);
  }).on("error", (e) => {
    callback(Error(e));
  });
};
```

En el siguiente ejemplo, la respuesta desde Amazon S3 se devolverá al invocador tan pronto como esté disponible. El tiempo de espera que se ejecuta en el bucle de eventos se ha detenido y continuará ejecutándose la próxima vez que se invoque la función.

### Note

Antes de usar este ejemplo, asegúrese de que el rol de ejecución de la función tenga permisos de lectura de Amazon S3.

## ES module handler

### Example – SDK de AWS v3 con `callbackWaitsForEmptyEventLoop`

En este ejemplo, se utiliza [AWS SDK for JavaScript v3](#), que está disponible en el tiempo de ejecución de `node.js 18.x` y posteriores.

```
import AWS from "@aws-sdk/client-s3";
const s3 = new AWS.S3({});

export const handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

## CommonJS module handler

### Example – SDK de AWS v3 con callbackWaitsForEmptyEventLoop

En este ejemplo, se utiliza [AWS SDK for JavaScript v3](#), que está disponible en el tiempo de ejecución de node.js 18.x y posteriores.

```
const AWS = require("@aws-sdk/client-s3");
const s3 = new AWS.S3({});

exports.handler = function (event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false;
  s3.listBuckets({}, callback);
  setTimeout(function () {
    console.log("Timeout complete.");
  }, 5000);
};
```

## Prácticas recomendadas de codificación para las funciones de Lambda en Node.js

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad. En Node.js, podría tener este aspecto:

```
exports.myHandler = function(event, context, callback) {
  var foo = event.foo;
  var bar = event.bar;
  var result = MyLambdaFunction (foo, bar);

  callback(null, result);
}

function MyLambdaFunction (foo, bar) {
  // MyLambdaFunction logic here
}
```

- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. Para los tiempos de ejecución de Node.js y Python,

estos incluyen los SDK de AWS. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.

- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación.
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.

- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Implementar funciones Node.js de Lambda con archivos de archivo.zip

El código de la función de AWS Lambda incluye un archivo .js o .mjs que contiene el código del controlador de la función, junto con los paquetes y módulos adicionales de los que depende el código. Para implementar este código de función en Lambda, se utiliza un paquete de despliegue. Este paquete puede ser un archivo de archivos .zip o una imagen de contenedor. Para obtener más información sobre el uso de imágenes de contenedores con Node.js, consulte [Implementar funciones de Lambda Node.js con imágenes de contenedores](#).

Para crear un paquete de despliegue como un archivo de archivos .zip, puede emplear utilidad de archivado de archivos .zip incorporada de la herramienta de línea de comandos, o cualquier otra utilidad de archivos .zip, como [7zip](#). En los ejemplos que se muestran en las siguientes secciones se supone que está utilizando una herramienta zip de línea de comandos en un entorno Linux o macOS. Para utilizar los mismos comandos en Windows, puede [instalar el Subsistema de Windows para Linux](#) a fin de obtener una versión de Ubuntu y Bash integrada con Windows.

Tenga en cuenta que Lambda utiliza permisos de archivo de POSIX, por lo que puede necesitar [establecer permisos para la carpeta del paquete de despliegue](#) antes de crear el archivo de archivos .zip.

## Temas

- [Dependencias de tiempo de ejecución en Node.js](#)
- [Creación de un paquete de despliegue .zip sin dependencias](#)
- [Creación de un paquete de despliegue .zip con dependencias](#)
- [Creación de una capa de Node.js para las dependencias](#)
- [Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución](#)
- [Creación y actualización de funciones de Lambda Node.js mediante archivos .zip](#)

## Dependencias de tiempo de ejecución en Node.js

Para las funciones de Lambda que utilizan el tiempo de ejecución de Node.js, una dependencia puede ser cualquier módulo Node.js. El tiempo de ejecución de Node.js incluye varias bibliotecas comunes, así como una versión de AWS SDK for JavaScript. El tiempo de ejecución nodejs16.x de Lambda incluye la versión 2.x del SDK. Las versiones de tiempo de ejecución nodejs18.x y las versiones posteriores incluyen la versión 3 del SDK. Para usar la versión 2 del SDK con las versiones



en tiempo de ejecución `nodejs18.x` y posteriores, agregue el SDK a su paquete de implementación de archivos `.zip`. Si el tiempo de ejecución elegido incluye la versión del SDK que utiliza, no necesita incluir la biblioteca del SDK en el archivo `.zip`. Para saber qué versión del SDK se incluye en el tiempo de ejecución que está utilizando, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Lambda actualiza de manera periódica las bibliotecas del SDK en el tiempo de ejecución de Node.js para incluir las características y actualizaciones de seguridad más recientes. Lambda también aplica parches y actualizaciones de seguridad a las demás bibliotecas incluidas en el tiempo de ejecución. Para tener el control total de las dependencias de un paquete, puede agregar la versión preferida de cualquier dependencia con tiempo de ejecución incluido al paquete de despliegue. Por ejemplo, si quiere utilizar una versión concreta del SDK para JavaScript, puede incluirla en el archivo `.zip` como una dependencia. Para obtener más información sobre cómo agregar dependencias con tiempo de ejecución incluido en el archivo `.zip`, consulte [Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución](#).

Según el [modelo de responsabilidad compartida de AWS](#), usted es responsable de la administración de cualquier dependencia en los paquetes de despliegue de sus funciones. Esto incluye la aplicación de actualizaciones y parches de seguridad. Para actualizar las dependencias del paquete de despliegue de la función, primero cree un nuevo archivo `.zip` y, a continuación, cárguelo en Lambda. Para obtener más información, consulte [Creación de un paquete de despliegue .zip con dependencias](#) y [Creación y actualización de funciones de Lambda Node.js mediante archivos .zip](#).

## Creación de un paquete de despliegue `.zip` sin dependencias

Si el código de la función no tiene dependencias, excepto las bibliotecas incluidas en el tiempo de ejecución de Lambda, el archivo `.zip` solo contiene el archivo `index.js` o `index.mjs` con el código del controlador de la función. Utilice la utilidad de compresión que prefiera para crear un archivo `.zip` con el archivo `index.js` o `index.mjs` en la raíz. Si el archivo que contiene el código del controlador no está en la raíz del archivo `.zip`, Lambda no podrá ejecutar el código.

Para obtener información sobre cómo implementar un archivo `.zip` para crear una nueva función de Lambda o actualizar una existente, consulte [Creación y actualización de funciones de Lambda Node.js mediante archivos .zip](#).

## Creación de un paquete de despliegue `.zip` con dependencias

Si el código de la función depende de paquetes o módulos que no se encuentran incluidos en el tiempo de ejecución de Node.js de Lambda, puede agregar estas dependencias al archivo `.zip` con el

código de la función o utilizar una [capa de Lambda](#). En las instrucciones de esta sección, se muestra cómo incluir las dependencias en el paquete de despliegue `.zip`. Para obtener instrucciones sobre cómo incluir las dependencias en una capa, consulte [the section called “Creación de una capa de Node.js para las dependencias”](#).

Los siguientes comandos de la CLI crean un archivo `.zip` llamado `my_deployment_package.zip`, que contiene el archivo `index.js` o `index.mjs` con el código del controlador de la función y sus dependencias. En el ejemplo, las dependencias se instalan mediante el administrador de paquetes `npm`.

### Creación del paquete de implementación

1. Desplácese hasta el directorio del proyecto que contiene el archivo de código fuente `index.js` o `index.mjs`. En este ejemplo, el directorio se llama `my_function`.

```
cd my_function
```

2. Instale las bibliotecas necesarias de la función en el directorio `node_modules` mediante el comando `npm install`. En este ejemplo, instalará AWS X-Ray SDK para Node.js.

```
npm install aws-xray-sdk
```

De este modo, se crea una estructura de carpetas similar a la siguiente:

```
~/my_function
### index.mjs
### node_modules
    ### async
    ### async-listener
    ### atomic-batcher
    ### aws-sdk
    ### aws-xray-sdk
    ### aws-xray-sdk-core
```

También puede agregar módulos personalizados que cree usted mismo al paquete de despliegue. Cree un directorio en `node_modules` con el nombre del módulo y guarde los paquetes personalizados allí.

3. Cree un archivo `.zip` con el contenido de la carpeta del proyecto en la raíz. Utilice la opción `r` (recursiva) para asegurarse de que el `zip` contiene las subcarpetas.

```
zip -r my_deployment_package.zip .
```

## Creación de una capa de Node.js para las dependencias

En las instrucciones de esta sección, se muestra cómo incluir las dependencias en una capa. Para obtener instrucciones sobre cómo incluir las dependencias en el paquete de implementación, consulte [the section called “Creación de un paquete de despliegue .zip con dependencias”](#).

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio /opt de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable PATH ya incluye rutas de carpeta específicas en el directorio /opt. Para garantizar que la variable PATH recoja el contenido de la capa, el archivo .zip de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

- nodejs/node\_modules
- nodejs/node16/node\_modules (NODE\_PATH)
- nodejs/node18/node\_modules (NODE\_PATH)
- nodejs/node20/node\_modules (NODE\_PATH)

Por ejemplo, la estructura del archivo .zip de la capa podría tener el siguiente aspecto:

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Además, Lambda detecta de forma automática cualquier biblioteca en el directorio /opt/lib y todos los archivos binarios en el directorio /opt/bin. Para asegurarse de que Lambda encuentre el contenido de la capa de forma correcta, también puede crear una capa con la siguiente estructura:

```
custom-layer.zip
# lib
  | lib_1
  | lib_2
# bin
  | bin_1
  | bin_2
```

Después de empaquetar la capa, consulte [the section called “Creación y eliminación de capas”](#) y [the section called “Adición de capas”](#) para completar la configuración de la capa.

## Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución

El tiempo de ejecución de Node.js incluye varias bibliotecas comunes, así como una versión de AWS SDK for JavaScript. Si quiere utilizar una versión diferente de una biblioteca con tiempo de ejecución incluido, puede agruparla con la función o agregarla como una dependencia en el paquete de despliegue. Por ejemplo, puede utilizar una versión diferente del SDK al agregarla al paquete de despliegue .zip. También puede incluirla en una [capa de Lambda](#) para la función.

Cuando utilice una instrucción `import` o `require` en el código, el tiempo de ejecución de Node.js buscará en los directorios, en la ruta `NODE_PATH`, hasta que encuentre el módulo. De forma predeterminada, la primera ubicación que busca el tiempo de ejecución es el directorio en el que se descomprime y monta el paquete de despliegue .zip (`/var/task`). Si incluye una versión de una biblioteca incluida en el tiempo de ejecución de su paquete de despliegue, esta versión tendrá prioridad sobre la versión incluida en el tiempo de ejecución. Las dependencias del paquete de despliegue también tienen prioridad sobre las dependencias de las capas.

Cuando agregue una dependencia a una capa, Lambda la extraerá en `/opt/nodejs/nodexx/node_modules`, donde `nodexx` representa la versión del entorno de ejecución utilizada. En la ruta de búsqueda, este directorio tiene prioridad sobre el directorio que contiene las bibliotecas incluidas en el tiempo de ejecución (`/var/lang/lib/node_modules`). Las bibliotecas de las capas de funciones tienen prioridad sobre las versiones incluidas en el tiempo de ejecución.

Para ver la ruta de búsqueda completa de la función de Lambda, agregue la siguiente línea de código.

```
console.log(process.env.NODE_PATH)
```

También puede agregar dependencias en una carpeta independiente dentro del paquete .zip. Por ejemplo, puede agregar un módulo personalizado a una carpeta del paquete .zip llamada `common`. Cuando descomprima y monte el paquete .zip, esta carpeta se colocará dentro del directorio `/var/task`. Para utilizar una dependencia de una carpeta en el paquete de despliegue .zip del código, utilice una instrucción `import { } from` o `const { } = require()`, según si utiliza una resolución de módulo CJS o ESM. Por ejemplo:

```
import { myModule } from './common'
```

Si agrupa el código con `esbuild`, `rollup` o similares, las dependencias utilizadas por la función se agrupan en uno o más archivos. Recomendamos utilizar este método para vender dependencias, siempre que sea posible. En comparación con el agregado de dependencias al paquete de despliegue, la agrupación del código mejora el rendimiento debido a la reducción de las operaciones de E/S.

## Creación y actualización de funciones de Lambda Node.js mediante archivos .zip

Una vez que haya creado su paquete de despliegue .zip, puede utilizarlo para crear una nueva función de Lambda o actualizar una existente. Puede implementar el paquete .zip a través de la consola, la AWS Command Line Interface y la API de Lambda. También puede crear y actualizar funciones de Lambda mediante AWS Serverless Application Model (AWS SAM) y AWS CloudFormation.

El tamaño máximo de un paquete de despliegue .zip para Lambda es de 250 MB (descomprimido). Tenga en cuenta que este límite se aplica al tamaño combinado de todos los archivos que cargue, incluidas las capas de Lambda.

El tiempo de ejecución de Lambda necesita permiso para leer los archivos del paquete de implementación. En la notación octal de permisos de Linux, Lambda necesita 644 permisos para archivos no ejecutables (`rw-r--r--`) y 755 permisos (`rw-r-xr-x`) para directorios y archivos ejecutables.

En Linux y macOS, utilice el comando `chmod` para cambiar los permisos de los archivos y directorios del paquete de implementación. Por ejemplo, para brindarle a un archivo ejecutable los permisos correctos, ejecute el siguiente comando.


```
chmod 755 <filepath>
```

Para cambiar los permisos de los archivos en Windows, consulte [Set, View, Change, or Remove Permissions on an Object](#) en la documentación de Microsoft Windows.

## Creación y actualización de funciones con archivos .zip mediante la consola

Para crear una nueva función, primero debe crearla en la consola y, a continuación, cargar el archivo .zip. Para actualizar una función existente, abra la página de la función correspondiente y, a continuación, siga el mismo procedimiento para agregar el archivo .zip actualizado.

Si el archivo .zip tiene un tamaño inferior a 50 MB, puede crear o actualizar una función al cargarlo directamente desde su equipo local. Para archivos .zip de más de 50 MB, primero debe cargar su paquete en un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS Management Console, consulte [Introducción a Amazon S3](#). Para cargar archivos mediante la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

 Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

Para crear una nueva función (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija Crear función.
2. Elija Crear desde cero.
3. En Información básica, haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función.
  - b. En Tiempo de ejecución, seleccione el tiempo de ejecución que desea utilizar.
  - c. (Opcional) Para Arquitectura, elija la arquitectura del conjunto de instrucciones para su función. La arquitectura predeterminada es x86\_64. Asegúrese de que el paquete de despliegue .zip para su función sea compatible con la arquitectura del conjunto de instrucciones que seleccione.
4. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o utilizar uno existente.
5. Elija Crear función. Lambda crea una función básica “Hola, mundo” mediante el tiempo de ejecución elegido.

Para cargar un archivo .zip desde su equipo local (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar el archivo .zip.
2. Seleccione la pestaña Código.

3. En el panel Código fuente, elija Cargar desde.
4. Elija un archivo .zip.
5. Para cargar el archivo .zip, haga lo siguiente:
  - a. Seleccione Cargar y, a continuación, seleccione su archivo .zip en el selector de archivos.
  - b. Elija Abrir.
  - c. Seleccione Guardar.

### Carga de un archivo .zip desde un bucket de Amazon S3 (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar un nuevo archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija la ubicación de Amazon S3.
5. Pegue la URL del enlace de Amazon S3 de su archivo .zip y seleccione Guardar.

### Actualización de las funciones del archivo .zip mediante el editor de código de la consola

Para algunas funciones con paquetes de despliegue en formato .zip, puede utilizar el editor de código integrado en la consola de Lambda para actualizar el código de la función de forma directa. Para utilizar esta característica, la función debe cumplir los siguientes criterios:

- La función debe utilizar uno de los tiempos de ejecución del lenguaje interpretado (Python, Node.js o Ruby)
- El paquete de despliegue de la función debe tener un tamaño inferior a 3 MB.

El código de función de las funciones con paquetes de despliegue de imágenes de contenedores no se puede editar directamente en la consola.

Para actualizar el código de función mediante el editor de código de la consola

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. Seleccione la pestaña Código.

3. En el panel Código fuente, seleccione su archivo de código fuente y edítelo en el editor de código integrado.
4. Cuando haya terminado de editar el código, seleccione Implementar para guardar los cambios y actualizar la función.

## Creación y actualización de funciones con archivos .zip mediante la AWS CLI

Puede utilizar la [AWS CLI](#) para crear una nueva función o actualizar una existente con un archivo .zip. Utilice los comandos [create-function](#) y [update-function-code](#) para implementar su paquete .zip. Si el archivo .zip tiene un tamaño inferior a 50 MB, puede cargarlo desde una ubicación de archivo en su equipo de compilación local. Para archivos más grandes, debe cargar su paquete .zip desde un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

### Note

Si carga su archivo .zip desde un bucket de Amazon S3 con la AWS CLI, el bucket debe estar ubicado en la misma Región de AWS que su función.

Para crear una nueva función mediante un archivo .zip con la AWS CLI, debe especificar lo siguiente:

- El nombre de la función (`--function-name`).
- El tiempo de ejecución de la función (`--runtime`).
- El nombre de recurso de Amazon (ARN) del [rol de ejecución](#) de la función (`--role`).
- El nombre del método de controlador en el código de la función (`--handler`).

También debe especificar la ubicación del archivo .zip. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda create-function --function-name myFunction \  
--runtime nodejs20.x --handler index.handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```



Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice la opción `--code`, como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `S3ObjectVersion` para los objetos con versiones.

```
aws lambda create-function --function-name myFunction \  
--runtime nodejs20.x --handler index.handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Para actualizar una función existente mediante la CLI, especifique el nombre de la función mediante el parámetro `--function-name`. También debe especificar la ubicación del archivo .zip que desea utilizar para actualizar el código de la función. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice las opciones `--s3-bucket` y `--s3-key` tal como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `--s3-object-version` para los objetos con versiones.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creación y actualización de funciones con archivos .zip mediante la API de Lambda

Para crear y actualizar funciones con un archivo de archivos .zip, utilice las siguientes operaciones de la API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creación y actualización de funciones con archivos .zip mediante AWS SAM

AWS Serverless Application Model (AWS SAM) es un conjunto de herramientas que ayuda a agilizar el proceso de creación y ejecución de aplicaciones sin servidor en AWS. Defina los recursos de

su aplicación en una plantilla YAML o JSON y utilice la interfaz de la línea de comandos de AWS SAM (AWS SAM CLI) para crear, empaquetar e implementar sus aplicaciones. Al crear una función de Lambda a partir de una plantilla de AWS SAM, AWS SAM crea automáticamente un paquete de despliegue .zip o una imagen de contenedor con el código de la función y las dependencias que especifique. Para obtener más información sobre el uso de AWS SAM para crear e implementar funciones de Lambda, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

También puede utilizar AWS SAM para crear una función de Lambda con un archivo de archivos .zip existente. Para crear una función de Lambda mediante AWS SAM, puede guardar el archivo .zip en un bucket de Amazon S3 o en una carpeta local de su equipo de compilación. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS SAM, el recurso `AWS::Serverless::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `CodeUri`: se establece como el URI de Amazon S3, la ruta a la carpeta local o el objeto [FunctionCode](#) del código de la función.
- `Runtime`: se establece como el entorno de ejecución elegido

Con AWS SAM, si su archivo .zip tiene más de 50 MB, no es necesario cargarlo primero en un bucket de Amazon S3. AWS SAM puede cargar paquetes .zip hasta el tamaño máximo permitido de 250 MB (descomprimidos) desde una ubicación de su equipo de compilación local.

Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS SAM.

## Creación y actualización de funciones con archivos .zip mediante AWS CloudFormation

Puede utilizar AWS CloudFormation para crear una función de Lambda con un archivo de archivos .zip. Para crear una función de Lambda a partir de un archivo .zip, primero debe cargar el archivo a un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS CloudFormation, el recurso `AWS::Lambda::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `Code`: ingrese el nombre del bucket de Amazon S3 y el nombre del archivo .zip en los campos `S3Bucket` y `S3Key`.
- `Runtime`: se establece como el tiempo de ejecución elegido.

El archivo .zip que genera AWS CloudFormation no puede superar los 4 MB. Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS CloudFormation, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

# Implementar funciones Node.js Lambda con imágenes de contenedor

Hay tres formas de crear una imagen de contenedor para una función de Lambda en Node.js:

- [Uso de una imagen base de AWS de Node.js](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

Las [imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También se pueden usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Node.js](#) en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Node.js](#) en la imagen.

## Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.

## Temas

- [AWS imágenes base para Node.js](#)
- [Uso de una imagen base de AWS de Node.js](#)
- [Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución](#)

## AWS imágenes base para Node.js

AWS proporciona las siguientes imágenes base para Node.js:

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
20	Node.js 20	Amazon Linux 2023	<a href="#">Dockerfile para Node.js 20 en GitHub</a>	
18	Node.js 18	Amazon Linux 2	<a href="#">Dockerfile para Node.js 18 en GitHub</a>	

Repositorio de Amazon ECR: [gallery.ecr.aws/lambda/nodejs](https://gallery.ecr.aws/lambda/nodejs)

Las imágenes base de Node.js 20 y versiones posteriores se basan en la [imagen de contenedor mínima de Amazon Linux 2023](#). Las imágenes base anteriores utilizan Amazon Linux 2. AL2023 ofrece varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como `glibc`.

Las imágenes basadas en AL2023 utilizan `microdnf` (enlazadas simbólicamente como `dnf`) como administrador de paquetes en lugar de `yum`, que es el administrador de paquetes predeterminado en Amazon Linux 2. `microdnf` es una implementación independiente de `dnf`. Para obtener una lista de los paquetes que se incluyen en las imágenes basadas en AL2023, consulte las columnas de Minimal Container de [Comparing packages installed on Amazon Linux 2023 Container Images](#). Para obtener más información sobre las diferencias entre AL2023 y Amazon Linux 2, consulte [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) en el Blog de informática de AWS.

**Note**

Para ejecutar imágenes basadas en AL2023 de forma local, incluso con AWS Serverless Application Model (AWS SAM), debe usar Docker en la versión 20.10.10 o posterior.

## Uso de una imagen base de AWS de Node.js

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#) (versión mínima 20.10.10 para las imágenes base de Node.js 20 y versiones posteriores)
- Node.js

### Creación de una imagen a partir de una imagen base

Para crear una imagen de contenedor a partir de una imagen base AWS para Node.js

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
cd example
```

2. Cree un nuevo proyecto de Node.js con npm. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

```
npm init
```

3. Cree un nuevo archivo denominado `index.js`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

### Example Controlador de CommonJS

```
exports.handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
```

```
};  
    return response;  
};
```

4. Si su función depende de bibliotecas distintas de AWS SDK for JavaScript, utilice [npm](#) para agregarlas al paquete.
5. Cree un nuevo archivo Dockerfile con la siguiente configuración:
  - Establezca la propiedad FROM en el [URI de la imagen base](#).
  - Utilice el comando COPY para copiar el código de la función y las dependencias del tiempo de ejecución a {LAMBDA\_TASK\_ROOT}, una [variable de entorno definido de Lambda](#).
  - Establezca el argumento CMD para el controlador de la función de Lambda.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:20  
  
# Copy function code  
COPY index.js ${LAMBDA_TASK_ROOT}  
  
# Set the CMD to your handler (could also be done as a parameter override outside  
of the Dockerfile)  
CMD [ "index.handler" ]
```

6. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen docker-image y se le asigna la [etiqueta](#) test.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

## (Opcional) Prueba de la imagen de forma local

1. Inicie la imagen de Docker con el comando `docker run`. En este ejemplo, `docker-image` es el nombre de la imagen y `test` es la etiqueta.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

2. Desde una nueva ventana de terminal, publique un evento en el punto de conexión local.

### Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{"payload": "hello world!"}'
```

### PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{}' -ContentType "application/json"
```



Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Obtenga el ID del contenedor.

```
docker ps
```

4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

**Note**

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

## Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución

Si usa una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir el cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución extiende el [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de la función.

Instale el [cliente de interfaz de tiempo de ejecución para Node.js](#) mediante el administrador de paquetes npm:

```
npm install aws-lambda-ric
```

También puede descargar el [cliente de interfaz de tiempo de ejecución Node.js](#) desde GitHub. El cliente de interfaz de tiempo de ejecución admite las siguientes versiones de Node.js:

- 14.x
- 16.x
- 18.x
- 20.x

El siguiente ejemplo muestra cómo crear una imagen de contenedor para Node.js utilizando una imagen base que no es de AWS. El siguiente Dockerfile de ejemplo usa una imagen base de `buster`. El Dockerfile incluye el cliente de interfaz de tiempo de ejecución.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#)
- Node.js

## Creación de imágenes a partir de una imagen base alternativa

Para crear una imagen de contenedor a partir de una imagen base que no es de AWS

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
cd example
```

2. Cree un nuevo proyecto de Node.js con npm. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

```
npm init
```

3. Cree un nuevo archivo denominado `index.js`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

### Example Controlador de CommonJS

```
exports.handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

4. Cree un nuevo Dockerfile. El siguiente Dockerfile usa una imagen base de `buster` en lugar de una [imagen base de AWS](#). El Dockerfile incluye el [cliente de interfaz de tiempo de ejecución](#), que hace que la imagen sea compatible con Lambda. El Dockerfile utiliza una [compilación de varias etapas](#). La primera etapa crea una imagen de compilación, que es un entorno estándar de Node.js donde se instalan las dependencias de la función. La segunda etapa crea una imagen más compacta que incluye el código de la función y sus dependencias. Esto reduce el tamaño final de la imagen.
  - Establezca la propiedad FROM como el identificador de la imagen base.

- Utilice el comando COPY para copiar el código de la función y las dependencias del tiempo de ejecución.
- Configure ENTRYPOINT como el módulo que desea que el contenedor de Docker ejecute cuando se inicie. En este caso, el módulo es el cliente de interfaz de tiempo de ejecución.
- Establezca el argumento CMD para el controlador de la función de Lambda.

## Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM node:20-buster as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Install build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

WORKDIR ${FUNCTION_DIR}

# Install Node.js dependencies
RUN npm install

# Install the runtime interface client
RUN npm install aws-lambda-ric

# Grab a fresh slim copy of the image to reduce the final size
FROM node:20-buster-slim

# Required for Node runtimes which use npm@8.6.0+ because
```

```
# by default npm writes logs under /home/.npm and Lambda fs is read-only
ENV NPM_CONFIG_CACHE=/tmp/.npm

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT ["/usr/local/bin/npx", "aws-lambda-ric"]
# Pass the name of the function handler as an argument to the runtime
CMD ["index.handler"]
```

5. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

#### (Opcional) Prueba de la imagen de forma local

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. Puede [crear el emulador en su imagen](#) o usar el procedimiento siguiente para instalarlo en su equipo local.

Para instalar y ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Desde el directorio del proyecto, ejecute el siguiente comando para descargar el emulador de interfaz de tiempo de ejecución (arquitectura x86-64) de GitHub e instalarlo en su equipo local.

## Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Para instalar el emulador arm64, reemplace la URL del repositorio de GitHub en el comando anterior por lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

## PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"  
if (-not (Test-Path $dirPath)) {  
  New-Item -Path $dirPath -ItemType Directory  
}  
  
$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/  
releases/latest/download/aws-lambda-rie"  
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"  
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Para instalar el emulador arm64, reemplace el `$downloadLink` con lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

2. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:

- `docker-image` es el nombre de la imagen y `test` es la etiqueta.
- `/usr/local/bin/npx aws-lambda-ric index.handler` es el ENTRYPOINT seguido del CMD de su Dockerfile.



## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  /usr/local/bin/npx aws-lambda-ric index.handler
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
  --entrypoint /aws-lambda/aws-lambda-rie `
  docker-image:test `
  /usr/local/bin/npx aws-lambda-ric index.handler
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

3. Publique un evento en el punto de conexión local.

## Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
'{"payload":"hello world!"}'
```

## PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
"application/json"
```

4. Obtenga el ID del contenedor.

```
docker ps
```

5. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace `3766c4ab331c` por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace `111122223333` por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:

- Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
- Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

# Uso del objeto de contexto Lambda para recuperar la información de la función Node.js

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución.

## Métodos de context

- `getRemainingTimeInMillis()`: devuelve el número de milisegundos que quedan antes del tiempo de espera de la ejecución.

## Propiedades de context

- `functionName`: el nombre de la función de Lambda.
- `functionVersion`: la [versión](#) de la función.
- `invokedFunctionArn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `memoryLimitInMB`: cantidad de memoria asignada a la función.
- `awsRequestId`: el identificador de la solicitud de invocación.
- `logGroupName`: grupo de registros de para la función.
- `logStreamName`: el flujo de registro de la instancia de la función.
- `identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
  - `cognitoIdentityId`: la identidad autenticada de Amazon Cognito.
  - `cognitoIdentityPoolId`: el grupo de identidad de Amazon Cognito que ha autorizado la invocación.
- `clientContext`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`

- `env.platform_version`
- `env.platform`
- `env.make`
- `env.model`
- `env.locale`
- Custom: valores personalizados que establece la aplicación del cliente.
- `callbackWaitsForEmptyEventLoop`: establézcalo en `false` para enviar la respuesta de forma inmediata cuando se ejecute la [devolución de llamada](#), en lugar de esperar a que el bucle de eventos de Node.js esté vacío. Si esto es `false`, los eventos pendientes siguen ejecutándose durante el siguiente periodo de invocación.

La siguiente función de ejemplo registra información de contexto y devuelve la ubicación de los registros.

#### Example Archivo `index.js`

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

# Registro y supervisión de las funciones de Lambda de Node.js

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre y envía registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación al flujo de registro y retransmite los registros y otras salidas desde el código de la función. Para obtener más información, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Esta página describe cómo producir resultados de registro a partir del código de la función de Lambda o registros de acceso mediante AWS Command Line Interface, la consola de Lambda o la consola de CloudWatch.

## Secciones

- [Crear una función que devuelve registros](#)
- [Uso de controles de registro avanzados de Lambda con Node.js](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)

## Crear una función que devuelve registros

Para generar registros desde el código de las funciones, puede utilizar los métodos del [objeto de la consola](#) o cualquier biblioteca de registro que escriba en `stdout` o en `stderr`. En el siguiente ejemplo, se registran los valores de las variables de entorno y el objeto de evento.

Example Archivo `index.js`: registro

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.info("EVENT\n" + JSON.stringify(event, null, 2))
  console.warn("Event not processed.")
  return context.logStreamName
}
```



## Example formato de registro

```
START RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Version: $LATEST
2019-06-07T19:11:20.562Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO ENVIRONMENT
VARIABLES
{
  "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
  "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/my-function",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2019/06/07/[$LATEST]e6f4a0c4241adcd70c262d34c0bbc85c",
  "AWS_EXECUTION_ENV": "AWS_Lambda_nodejs12.x",
  "AWS_LAMBDA_FUNCTION_NAME": "my-function",
  "PATH": "/var/lang/bin:/usr/local/bin:/usr/bin/::bin:/opt/bin",
  "NODE_PATH": "/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/
node_modules",
  ...
}
2019-06-07T19:11:20.563Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO EVENT
{
  "key": "value"
}
2019-06-07T19:11:20.564Z c793869b-ee49-115b-a5b6-4fd21e8dedac WARN Event not processed.
END RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac
REPORT RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Duration: 128.83 ms Billed
Duration: 200 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms
XRAY TraceId: 1-5d9d007f-0a8c7fd02xmpl1480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled:
true
```

El tiempo de ejecución de Node.js registra las líneas START, END y REPORT de cada invocación. Se agrega una marca de tiempo, un ID de solicitud y el nivel de registro en cada entrada registrada por la función. La línea del informe proporciona los siguientes detalles.

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.

- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- Traceld de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

Puede ver los registros en la consola de Lambda o en la de Registros de CloudWatch, o bien en la línea de comandos.

## Uso de controles de registro avanzados de Lambda con Node.js

Para tener más control sobre cómo se capturan, procesan y consumen los registros de sus funciones, puede configurar las siguientes opciones de registro para los tiempos de ejecución de Node.js compatibles:

- Formato de registro: seleccione entre texto sin formato y el formato JSON estructurado para los registros de su función
- Nivel de registro: para los registros en formato JSON, elija el nivel de detalle de los registros que Lambda envía a Amazon CloudWatch, como ERROR, DEBUG o INFO
- Grupo de registro: elija el grupo de registro de CloudWatch al que su función envía los registros

Para obtener más información sobre estas opciones de registro e instrucciones sobre cómo configurar la función para utilizarlas, consulte [the section called “Configuración de registros de funciones”](#).

Para usar las opciones de formato y nivel de registro con las funciones de Lambda de Node.js, consulte las instrucciones de las siguientes secciones.

## Uso de registros JSON estructurados con Node.js

Si selecciona JSON para el formato de registro de la función, Lambda enviará los registros que obtenga mediante los métodos consola `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` y `console.warn` a CloudWatch como JSON estructurado. Cada objeto de registro JSON contiene, por lo menos, cuatro pares clave-valor con las siguientes claves:

- "timestamp": la hora en que se generó el mensaje de registro

- "level": el nivel de registro asignado al mensaje
- "message": el contenido del mensaje de registro
- "requestId": el ID de solicitud único para la invocación de la función

Según el método de registro que use la función, este objeto JSON también puede contener pares de claves adicionales. Por ejemplo, si la función usa métodos `console` para registrar objetos de error con varios argumentos, el objeto JSON contendrá pares clave-valor adicionales junto con las claves `errorMessage`, `errorType`, y `stackTrace`.

Si su código ya usa otra biblioteca de registro, como Powertools para AWS Lambda, para producir registros JSON estructurados, no necesita realizar ningún cambio. Lambda no codifica dos veces ningún registro que ya esté codificado en JSON, por lo que los registros de la aplicación de su función seguirán capturándose como antes.

Para obtener más información sobre el uso del paquete de registro de Powertools para AWS Lambda a fin de crear registros JSON estructurados en el tiempo de ejecución de Node.js, consulte [the section called "Registro"](#).

### Ejemplo de salidas de registro con formato JSON

En los siguientes ejemplos, se muestra cómo se capturan en Registros de CloudWatch varias salidas de registro generadas con los métodos `console` con argumentos únicos y múltiples cuando se establece el formato de registro de la función en JSON.

En el primer ejemplo, se usa el método `console.error` para generar una cadena sencilla.

#### Example Código de registro de Node.js

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
}
```

#### Example Entrada de registro JSON

```
{
  "timestamp": "2023-11-01T00:21:51.358Z",
  "level": "ERROR",
```

```
"message": "This is a warning message",
"requestId": "93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

También se pueden generar mensajes de registro estructurados de manera más compleja mediante argumentos únicos o múltiples con los métodos `console`. En el siguiente ejemplo, se usa `console.log` para generar dos pares clave-valor con un único argumento. Tenga en cuenta que el campo "message" del objeto JSON que Lambda envía a Registros de CloudWatch no se representa en forma de cadena.

#### Example Código de registro de Node.js

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

#### Example Entrada de registro JSON

```
{
  "timestamp": "2023-12-08T23:21:04.664Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": {
    "data": 12.3,
    "flag": false
  }
}
```

En el siguiente ejemplo, se volverá a usar el método `console.log` para crear una salida de registro. Esta vez, el método utiliza dos argumentos: un mapa que contiene dos pares clave-valor y una cadena de identificación. Tenga en cuenta que, en este caso, dado que se proporcionaron dos argumentos, Lambda representa el campo "message" en forma de cadena.

#### Example Código de registro de Node.js

```
export const handler = async (event) => {
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

## Example Entrada de registro JSON

```
{
  "timestamp": "2023-12-08T23:21:04.664Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": "Some object - { data: 12.3, flag: false }"
}
```

Lambda asigna a las salidas generadas mediante `console.log` el nivel de registro INFO.

En el último ejemplo, se muestra cómo los objetos de error se pueden enviar a Registros de CloudWatch mediante los métodos `console`. Tenga en cuenta que, cuando registra objetos de error con varios argumentos, Lambda agrega los campos `errorMessage`, `errorType` y `stackTrace` a la salida del registro.

## Example Código de registro de Node.js

```
export const handler = async (event) => {
  let e1 = new ReferenceError("some reference error");
  let e2 = new SyntaxError("some syntax error");
  console.log(e1);
  console.log("errors logged - ", e1, e2);
};
```

## Example Entrada de registro JSON

```
{
  "timestamp": "2023-12-08T23:21:04.632Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": {
    "errorType": "ReferenceError",
    "errorMessage": "some reference error",
    "stackTrace": [
      "ReferenceError: some reference error",
      "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
      "    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"
    ]
  }
}
```

```
{
  "timestamp": "2023-12-08T23:21:04.646Z",
  "level": "INFO",
  "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
  "message": "errors logged - ReferenceError: some reference error
\n    at Runtime.handler (file:///var/task/index.mjs:3:12)\n    at
Runtime.handleOnceNonStreaming
  (file:///var/runtime/index.mjs:1173:29) SyntaxError: some syntax
error\n    at Runtime.handler (file:///var/task/index.mjs:4:12)\n    at
Runtime.handleOnceNonStreaming
  (file:///var/runtime/index.mjs:1173:29)",
  "errorType": "ReferenceError",
  "errorMessage": "some reference error",
  "stackTrace": [
    "ReferenceError: some reference error",
    "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
    "    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"
  ]
}
```

Al registrar varios tipos de error, los campos adicionales `errorMessage`, `errorType` y `stackTrace` se extraen del primer tipo de error suministrado al método `console`.

## Uso de bibliotecas cliente de formato de métricas integradas (EMF) con registros JSON estructurados

AWS proporciona bibliotecas cliente de código abierto para Node.js que puede utilizar para crear registros de [formato de métricas integradas](#) (EMF). Si tiene ya funciones que utilizan estas bibliotecas y cambia el formato de registro de la función a JSON, es posible que CloudWatch deje de reconocer las métricas emitidas por el código.

Si, en este momento, su código emite registros EMF directamente con `console.log` o `PowerTools` para AWS Lambda (TypeScript), CloudWatch tampoco podrá analizarlos si cambia el formato de registro de la función a JSON.

### Important

Para asegurarse de que CloudWatch siga analizando correctamente los registros de EMF de sus funciones, actualice sus bibliotecas de [EMF](#) y [PowerTools para AWS Lambda](#) a las versiones más recientes. Si cambia al formato de registro JSON, también le recomendamos

que realice pruebas para garantizar la compatibilidad con las métricas integradas de su función. Si su código emite registros de EMF directamente con `console.log`, cámbielo para que genere esas métricas de forma directa a `stdout`, como se muestra en el siguiente ejemplo de código.

### Example Código que emite métricas integradas a **stdout**

```
process.stdout.write(JSON.stringify(
  {
    "_aws": {
      "Timestamp": Date.now(),
      "CloudWatchMetrics": [{
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [{
          "Name": "time",
          "Unit": "Milliseconds",
          "StorageResolution": 60
        }]
      }]
    },
    "functionVersion": "$LATEST",
    "time": 100,
    "requestId": context.awsRequestId
  }
) + "\n")
```

### Uso del filtrado a nivel de registro con Node.js

Para que AWS Lambda filtre los registros de las aplicaciones según su nivel de registro, la función debe usar registros con formato JSON. Puede lograr esto de dos maneras:

- Cree salidas de registro con los métodos consola estándar y configure su función para que utilice el formato de registro JSON. A continuación, AWS Lambda filtra las salidas de registro con el par clave-valor “nivel” del objeto JSON descrito en [the section called “Uso de registros JSON estructurados con Node.js”](#). Para obtener información sobre cómo configurar el formato de registro de la función, consulte [the section called “Configuración de registros de funciones”](#).
- Utilice otra biblioteca o método de registro para crear registros estructurados en JSON en su código que incluyan un par clave-valor “nivel” que defina el nivel de la salida del registro. Por

ejemplo, puede utilizar Powertools para AWS Lambda con el objetivo de generar salidas de registros JSON estructurados a partir de su código. Consulte [the section called “Registro”](#) para obtener más información sobre el uso de Powertools con el tiempo de ejecución de Node.js.

Para que Lambda filtre los registros de la función, también debe incluir un par clave-valor "timestamp" en la salida del registro JSON. La hora debe especificarse con un formato de marca de tiempo [RFC 3339](#) válido. Si no proporciona una marca de tiempo válida, Lambda asignará al registro el nivel INFO y agregará una marca de tiempo por usted.

Cuando configura la función para que utilice el filtrado a nivel de registro, selecciona el nivel de registros que desea que AWS Lambda envíe a Registros de Amazon CloudWatch de las siguientes opciones:

Nivel de registro	Uso estándar
TRACE (más detallado)	La información más detallada que se utiliza para rastrear la ruta de ejecución del código
DEBUG	Información detallada para la depuración del sistema
INFO	Mensajes que registran el funcionamiento normal de su función
WARN	Mensajes sobre posibles errores que pueden provocar un comportamiento inesperado si no se abordan
ERROR	Mensajes sobre problemas que impiden que el código funcione según lo esperado
FATAL (menos detallado)	Mensajes sobre errores graves que hacen que la aplicación deje de funcionar

Lambda envía los registros del nivel seleccionado y en un nivel inferior a CloudWatch. Por ejemplo, si configura un nivel de registro de WARN, Lambda enviará los registros correspondientes a los niveles WARN, ERROR y FATAL.



## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

### Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBu1QgUmVxdWVzdElk0iA4N2QwNDRi0C1mMTU0LTExZTgt0GNkYS0y0Tc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

### Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
```

```
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad base64 está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar base64 -D.

### Example get-logs.sh script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza sed para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando get-log-events.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como get-logs.sh.

La opción cli-binary-format es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute aws configure set cli-binary-format raw-in-base64-out. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

### Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
}
```

```
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"  
}
```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

# Instrumentación del código Node.js en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas dos bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK para Node.js](#): un SDK para generar y enviar datos de seguimiento a X-Ray.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de ADOT para instrumentar las funciones de Node.js](#)
- [Uso del SDK de X-Ray para instrumentar las funciones de Node.js](#)
- [Activación del seguimiento con la consola de Lambda](#)
- [Activación del seguimiento con la API de Lambda](#)
- [Activación del seguimiento con AWS CloudFormation](#)
- [Interpretación de un seguimiento de X-Ray](#)
- [Almacenamiento de dependencias de tiempo de ejecución en una capa \(X-Ray SDK\)](#)

## Uso de ADOT para instrumentar las funciones de Node.js

ADOT proporciona [capas](#) de Lambda completamente administradas que empaquetan todo lo necesario para recopilar datos de telemetría mediante el OTel SDK. Utilizando esta capa, se pueden instrumentar las funciones de Lambda sin tener que modificar el código de ninguna función. También se puede configurar la capa para que realice una inicialización personalizada de OTel. Para obtener más información, consulte [Configuración personalizada del compilador de ADOT en Lambda](#) en la documentación de ADOT.

Para los tiempos de ejecución de Node.js, puede agregar la capa de Lambda administrada por AWS para ADOT Javascript a fin de instrumentar de forma automática las funciones. Para obtener instrucciones detalladas sobre cómo agregar esta capa, consulte [Soporte de Lambda de AWS Distro for OpenTelemetry para JavaScript](#) en la documentación de ADOT.

## Uso del SDK de X-Ray para instrumentar las funciones de Node.js

Para registrar detalles sobre las llamadas que realiza la función Lambda a otros recursos de la aplicación, también se puede utilizar el AWS X-Ray SDK para Node.js. Para obtener el SDK, agregue el paquete `aws-xray-sdk-core` a las dependencias de la aplicación.

Example [blank-nodejs/package.json](#)

```
{
  "name": "blank-nodejs",
  "version": "1.0.0",
  "private": true,
  "devDependencies": {
    "jest": "29.7.0"
  },
  "dependencies": {
    "@aws-sdk/client-lambda": "3.345.0",
    "aws-xray-sdk-core": "3.5.3"
  },
  "scripts": {
    "test": "jest"
  }
}
```

Para instrumentar los clientes del AWS SDK en la [AWS SDK for JavaScript versión 3](#), empaquete la instancia del cliente con el método `captureAWSv3Client`.

Example [blank-nodejs/function/index.js](#): seguimiento de un cliente SDK de AWS.

```
const AWSXRay = require('aws-xray-sdk-core');
const { LambdaClient, GetAccountSettingsCommand } = require('@aws-sdk/client-lambda');

// Create client outside of handler to reuse
const lambda = AWSXRay.captureAWSv3Client(new LambdaClient());

// Handler
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    ...
```

El tiempo de ejecución de Lambda establece algunas variables de entorno para configurar el SDK de X-Ray. Por ejemplo, Lambda establece `AWS_XRAY_CONTEXT_MISSING` para `LOG_ERROR` a fin de evitar arrojar errores de tiempo de ejecución desde el SDK de X-Ray. Para establecer una estrategia de falta de contexto personalizada, invalide la variable de entorno en la configuración de la función para que no tenga valor y, a continuación, puede establecer la estrategia de falta de contexto mediante programación.

Example Ejemplo de código de inicialización

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Para obtener más información, consulte [the section called “Configuración de variables de entorno”](#).

Una vez agregadas las dependencias correctas y realizados los cambios de código necesarios, active el seguimiento en la configuración de la función mediante la consola de Lambda o la API.

## Activación del seguimiento con la consola de Lambda

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.



3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

## Activación del seguimiento con la API de Lambda

Configure el rastreo en la función Lambda con AWS CLI o SDK de AWS, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Activación del seguimiento con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM) , utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

## Interpretación de un seguimiento de X-Ray

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

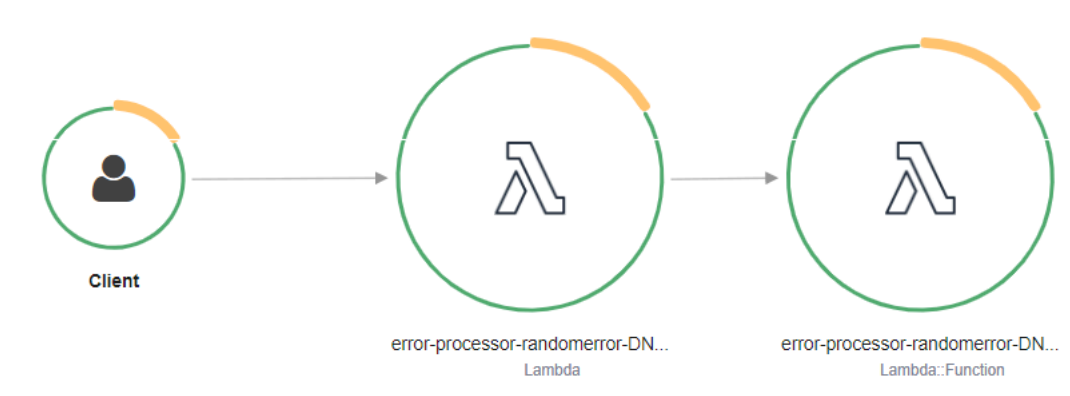
Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.



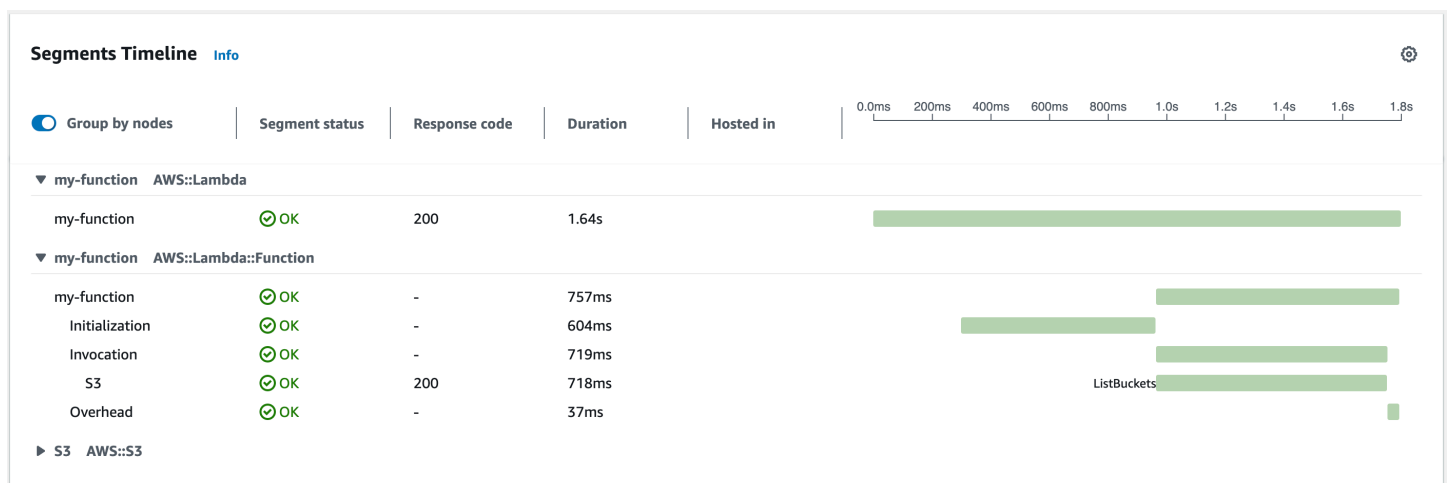
X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra

representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo, se muestra un seguimiento con estos dos segmentos. Ambos se denominan my-function, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.
- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte [AWS X-Ray SDK para Node.js](#) en la Guía para desarrolladores de AWS X-Ray.

### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza

cargos por almacenamiento y recuperación del seguimiento. Para más información, consulte [Precios de AWS X-Ray](#).

## Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK)

Si utiliza el X-Ray SDK para instrumentar el código de las funciones de los clientes del SDK de AWS, el paquete de implementación puede llegar a ser bastante grande. Para evitar que se carguen dependencias en tiempo de ejecución cada vez que se actualice el código de las funciones, empaquete el X-Ray SDK en una [capa de Lambda](#).

El siguiente ejemplo muestra un recurso `AWS::Serverless::LayerVersion` que almacena el AWS X-Ray SDK para Node.js.

Example [template.yml](#): capa de dependencias.

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - nodejs16.x
```

Con esta configuración, solo se actualiza la capa de la biblioteca si se modifican las dependencias del tiempo de ejecución. Dado que el paquete de implementación de la función contiene únicamente el código, esto puede ayudar a reducir los tiempos de carga.

Para crear una capa de dependencias, es necesario realizar cambios en la compilación para generar el archivo de capas antes de la implementación. Para ver un ejemplo de trabajo, consulte la aplicación de ejemplo [blank-nodejs](#) .

# Creación de funciones de Lambda con TypeScript

Se puede utilizar el tiempo de ejecución de Node.js para ejecutar código TypeScript en AWS Lambda. Dado que Node.js no ejecuta código de TypeScript de manera nativa, antes se debe transpilar el código de TypeScript a JavaScript. A continuación, utilice los archivos JavaScript para implementar el código de la función en Lambda. El código se ejecuta en un entorno que incluye AWS SDK para JavaScript, con credenciales de un rol de AWS Identity and Access Management (IAM) que usted administra. Para obtener más información sobre las versiones del SDK incluidas en los tiempos de ejecución de Node.js, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Lambda admite los siguientes tiempos de ejecución de Node.js.

## Node.js

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Node.js 20	nodejs20.x	Amazon Linux 2023			
Node.js 18	nodejs18.x	Amazon Linux 2			

## Temas

- [Configuración de un entorno de desarrollo de TypeScript](#)
- [Definir el controlador de funciones de Lambda en Typescript](#)
- [Implemente TypeScript código transpilado en Lambda con archivos de archivos.zip](#)
- [Implementar código de TypeScript transpilado en Lambda con imágenes de contenedor](#)
- [Uso del objeto de contexto Lambda para recuperar la información de la función de TypeScript](#)
- [Registro y supervisión de las funciones de Lambda de TypeScript](#)
- [Seguimiento del código de TypeScript en AWS Lambda](#)

## Configuración de un entorno de desarrollo de TypeScript

Utilice un entorno de desarrollo integrado (IDE) local, un editor de texto o [AWS Cloud9](#) para escribir el código de la función de TypeScript. No se puede crear código de TypeScript en la consola de Lambda.

Para transpilar el código de TypeScript, configure un compilador, como [esbuild](#) o el compilador de TypeScript de Microsoft (`tsc`), que se incluye con la [distribución de TypeScript](#). Puede utilizar [AWS Serverless Application Model \(AWS SAM\)](#) o [AWS Cloud Development Kit \(AWS CDK\)](#) para simplificar la compilación e implementación de código de TypeScript. Ambas herramientas utilizan `esbuild` para transpilar código de TypeScript a JavaScript.

Al utilizar `esbuild`, tenga en cuenta lo siguiente:

- Existen varias [advertencias en relación con TypeScript](#).
- La configuración de transpilación de TypeScript se debe establecer de modo que coincida con el tiempo de ejecución de Node.js que se piense utilizar. Para obtener más información, consulte [Target](#) (Destino) en la documentación de `esbuild`. Para ver un ejemplo de archivo `tsconfig.json` que muestra cómo orientarse a una versión específica de Node.js compatible con Lambda, consulte el [repositorio de GitHub de TypeScript](#).
- `esbuild` no realiza comprobaciones de tipos. Para comprobar los tipos, utilice el compilador `tsc`. Ejecute `tsc -noEmit` o agregue un parámetro `"noEmit"` al archivo `tsconfig.json`, como se muestra en el siguiente ejemplo. Eso configura `tsc` para que no emita archivos JavaScript. Después de comprobar los tipos, utilice `esbuild` para convertir los archivos de TypeScript a JavaScript.

### Example `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "es2020",
    "strict": true,
    "preserveConstEnums": true,
    "noEmit": true,
    "sourceMap": false,
    "module": "commonjs",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "skipLibCheck": true,
```



```
    "forceConsistentCasingInFileNames": true,  
    "isolatedModules": true,  
  },  
  "exclude": ["node_modules", "**/*.test.ts"]  
}
```

# Definir el controlador de funciones de Lambda en Typescript

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

## Temas

- [Conceptos básicos del controlador de Typescript](#)
- [Uso de async/await](#)
- [Uso de devolución de llamadas](#)
- [Uso de tipos para el objeto event](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Typescript](#)

## Conceptos básicos del controlador de Typescript

### Example Controlador de TypeScript

Esta función de ejemplo registra el contenido del objeto de evento y devuelve la ubicación de los registros. Tenga en cuenta lo siguiente:

- Antes de utilizar este código en una función de Lambda, debe agregar el paquete [@types/aws-lambda](#) como dependencia de implementación. Este paquete contiene las definiciones de tipos de Lambda. Cuando `@types/aws-lambda` se encuentra instalado, la instrucción `import (import ... from 'aws-lambda')` importa las definiciones de tipo. No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio de GitHub DefinitelyTyped.
- El controlador de este ejemplo es un módulo ES y debe designarse como tal en el archivo `package.json` o mediante la extensión de archivo `.mjs`. Para obtener más información, consulte [Designación de un controlador de funciones como módulo de ES](#).

```
import { Handler } from 'aws-lambda';

export const handler: Handler = async (event, context) => {
  console.log('EVENT: \n' + JSON.stringify(event, null, 2));
  return context.logStreamName;
};
```

El tiempo de ejecución pasa argumentos al método del controlador. El primer argumento es el objeto `event`, que contiene información del invocador. El invocador pasa esta información como una cadena con formato JSON cuando llama a [Invoke](#) y el runtime la convierte en un objeto. Cuando un servicio de AWS invoca su función, la estructura del evento [varía en función del servicio](#). Con TypeScript, recomendamos utilizar anotaciones de texto para el objeto de evento. Para obtener más información, consulte [Uso de tipos para el objeto event](#).

El segundo argumento es el [objeto context](#), que incluye información sobre la invocación, la función y el entorno de ejecución. En el ejemplo anterior, la función obtiene el nombre de [flujo de registro](#) del objeto context y lo devuelve al invocador.

También puede utilizar un argumento de devolución de llamada, una función a la que puede llamar en los controladores non-async para enviar una respuesta. Le recomendamos que utilice `async/await` en lugar de devolución de llamadas. `Async/await` mejora la legibilidad, el manejo de errores y la eficiencia. Para obtener información sobre las diferencias entre `async/await` y las devoluciones de llamadas, consulte [Uso de devolución de llamadas](#).

## Uso de `async/await`

Si su código realiza una tarea asíncrona, utilice el patrón `async/await` para asegurarse de que el controlador termina de ejecutarse. `Async/await` es una forma concisa y legible de escribir código asíncrono en Node.js, sin necesidad de devoluciones de llamadas anidadas ni promesas encadenadas. Con `async/await`, puedes escribir código que se lea como código sincrónico, sin dejar de ser asíncrono y sin bloqueo.

La palabra clave `async` marca una función como asíncrona y la palabra clave `await` detiene la ejecución de la función hasta que se resuelva una `Promise`.

Example Función de TypeScript: asíncrona

En este ejemplo se utiliza `fetch`, que está disponible en el tiempo de ejecución de `node.js 18.x`. Tenga en cuenta lo siguiente:

- Antes de utilizar este código en una función de Lambda, debe agregar el paquete [@types/aws-lambda](#) como dependencia de implementación. Este paquete contiene las definiciones de tipos de Lambda. Cuando `@types/aws-lambda` se encuentra instalado, la instrucción `import (import ... from 'aws-lambda')` importa las definiciones de tipo. No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio de GitHub DefinitelyTyped.

- El controlador de este ejemplo es un módulo ES y debe designarse como tal en el archivo `package.json` o mediante la extensión de archivo `.mjs`. Para obtener más información, consulte [Designación de un controlador de funciones como módulo de ES](#).

```
import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
const url = 'https://aws.amazon.com/';
export const lambdaHandler = async (event: APIGatewayProxyEvent):
  Promise<APIGatewayProxyResult> => {
  try {
    // fetch is available with Node.js 18
    const res = await fetch(url);
    return {
      statusCode: res.status,
      body: JSON.stringify({
        message: await res.text(),
      }),
    };
  } catch (err) {
    console.log(err);
    return {
      statusCode: 500,
      body: JSON.stringify({
        message: 'some error happened',
      }),
    };
  }
};
```

## Uso de devolución de llamadas

Le recomendamos que utilice [async/await](#) para declarar el controlador de funciones en lugar de utilizar devolución de llamadas. Async/await es una mejor opción por diversas razones:

- Legibilidad: el código `async/await` es más fácil de leer y entender que el código de devolución de llamada, que puede volverse difícil de seguir con rapidez y provocar un caos de devolución de llamadas.
- Depuración y gestión de errores: depurar código basado en devoluciones de llamadas puede resultar difícil. La pila de llamadas puede resultar difícil de seguir y los errores se pueden pasar por alto con facilidad. Con `async/await`, puede utilizar bloques `try/catch` para gestionar los errores.

- **Eficiencia:** las devoluciones de llamadas a menudo requieren cambiar entre diferentes partes del código. `Async/await` puede reducir la cantidad de cambios de contexto, lo que resulta en un código más eficiente.

Cuando utiliza devoluciones de llamadas en su controlador, la función prosigue con su ejecución hasta que el [bucle de eventos](#) está vacío o se agota el tiempo de espera de la función. La respuesta no se envía al invoker hasta que todas las tareas de bucle de eventos hayan terminado. Si el tiempo de espera de la función se agota, se devolverá un error en su lugar. Puede configurar el tiempo de ejecución para enviar la respuesta inmediatamente estableciendo [`context.callbackWaitsForEmptyEventLoop`](#) en `false`.

La función de devolución de llamada toma dos argumentos: un `Error` y una respuesta. El objeto de respuesta debe ser compatible con `JSON.stringify`.

#### Example Función de TypeScript con devolución de llamada

Esta función de muestra recibe un evento de Amazon API Gateway, registra los objetos de evento y contexto y luego devuelve una respuesta a API Gateway. Tenga en cuenta lo siguiente:

- Antes de utilizar este código en una función de Lambda, debe agregar el paquete [@types/aws-lambda](#) como dependencia de implementación. Este paquete contiene las definiciones de tipos de Lambda. Cuando `@types/aws-lambda` se encuentra instalado, la instrucción `import (import ... from 'aws-lambda')` importa las definiciones de tipo. No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio de GitHub DefinitelyTyped.
- El controlador de este ejemplo es un módulo ES y debe designarse como tal en el archivo `package.json` o mediante la extensión de archivo `.mjs`. Para obtener más información, consulte [Designación de un controlador de funciones como módulo de ES](#).

```
import { Context, APIGatewayProxyCallback, APIGatewayEvent } from 'aws-lambda';

export const lambdaHandler = (event: APIGatewayEvent, context: Context, callback:
  APIGatewayProxyCallback): void => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  callback(null, {
    statusCode: 200,
    body: JSON.stringify({
```

```
        message: 'hello world',
    })),
  });
};
```

## Uso de tipos para el objeto event

Se recomienda no utilizar el tipo [any](#) para los argumentos del controlador y el tipo de devolución, porque se pierde la capacidad de comprobar los tipos. En su lugar, genere un evento mediante el comando [sam local generate-event](#) de la CLI de AWS Serverless Application Model, o bien utilice una definición de código abierto del [paquete @types/aws-lambda](#).

Generación de un evento mediante el comando `sam local generate-event`

1. Genere un evento de proxy de Amazon Simple Storage Service (Amazon S3).

```
sam local generate-event s3 put >> S3PutEvent.json
```

2. Utilice la [utilidad quicktype](#) para generar definiciones de tipos a partir del archivo `S3putevent.json`.

```
npm install -g quicktype
quicktype S3PutEvent.json -o S3PutEvent.ts
```

3. Utilice los tipos generados en el código.

```
import { S3PutEvent } from './S3PutEvent';

export const lambdaHandler = async (event: S3PutEvent): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

Generación de un evento mediante una definición de código abierto del paquete [@types/aws-lambda](#)

1. Agregue el paquete [@types/aws-lambda](#) en forma de dependencia de desarrollo.

```
npm install -D @types/aws-lambda
```

2. Utilice los tipos en el código.

```
import { S3Event } from "aws-lambda";

export const lambdaHandler = async (event: S3Event): Promise<void> => {
  event.Records.map((record) => console.log(record.s3.object.key));
};
```

## Prácticas recomendadas de codificación para las funciones de Lambda en Typescript

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad. En Node.js, podría tener este aspecto:

```
exports.myHandler = function(event, context, callback) {
  var foo = event.foo;
  var bar = event.bar;
  var result = MyLambdaFunction (foo, bar);

  callback(null, result);
}

function MyLambdaFunction (foo, bar) {
  // MyLambdaFunction logic here
}
```

- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. Para los tiempos de ejecución de Node.js y Python, estos incluyen los SDK de AWS. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).

- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación.
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.



- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Implemente TypeScript código transpilado en Lambda con archivos de archivos.zip

Antes de poder implementar el TypeScript código en AWS Lambda, debe transpirarlo en JavaScript. En esta página se explican tres formas de crear e implementar TypeScript código en Lambda con archivos.zip:

- [Uso de AWS Serverless Application Model \(AWS SAM\)](#)
- [Con AWS Cloud Development Kit \(AWS CDK\)](#)
- [Uso de AWS Command Line Interface \(AWS CLI\) y esbuild](#)

AWS SAM y AWS CDK simplifique la creación y el despliegue de funciones TypeScript. La [especificación de plantillas de AWS SAM](#) proporciona una sintaxis sencilla y limpia para describir las funciones, las API, los permisos, las configuraciones y los eventos de Lambda que constituyen la aplicación sin servidor. [AWS CDK](#) permite crear aplicaciones fiables, escalables y rentables en la nube con la considerable potencia expresiva de un lenguaje de programación. AWS CDK se dirige a usuarios de AWS con una experiencia entre media y alta. AWS CDK tanto él como el AWS SAM utilizan esbuild para transpirar el código. TypeScript JavaScript

## Uso AWS SAM para implementar TypeScript código en Lambda

Siga los pasos que se indican a continuación para descargar, crear e implementar un ejemplo de TypeScript aplicación Hello World mediante AWS SAM. Esta aplicación implementa un backend de API básico. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, se invoca la función de Lambda. La función devuelve el mensaje `hello world`.

### Note

AWS SAM usa esbuild para crear funciones Lambda de Node.js a TypeScript partir del código. El soporte de esbuild se encuentra actualmente en versión preliminar pública. Mientras se encuentre en versión preliminar pública, el soporte de esbuild puede verse sometido a cambios incompatibles con versiones anteriores.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#)
- Node.js 18.x

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla Hello World. TypeScript

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs18.x
```

2. (Opcional) La aplicación de ejemplo incluye configuraciones para herramientas de uso habitual, tales como [ESLint](#) para linting de código y [Jest](#) para pruebas unitarias. Para ejecutar comandos de lint y pruebas:

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Compile la aplicación.

```
cd sam-app
sam build
```

4. Implemente la aplicación.

```
sam deploy --guided
```

5. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, responda con Enter.
6. El resultado muestra el punto de conexión de la API REST. Abra el punto de conexión en un navegador para probar la función. Debería ver esta respuesta:

```
{"message":"hello world"}
```

7. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Uso del AWS CDK para implementar TypeScript código en Lambda

Siga los pasos que se indican a continuación para crear e implementar una TypeScript aplicación de ejemplo mediante AWS CDK. Esta aplicación implementa un backend de API básico. Consiste en un punto de conexión de API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, se invoca la función de Lambda. La función devuelve el mensaje `hello world`.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- Node.js 18.x
- Bien sea [Docker](#) o [esbuild](#)

### Implementar un aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world  
cd hello-world
```

2. Inicialice la aplicación.

```
cdk init app --language typescript
```

3. Agregue el paquete [@types/aws-lambda](#) en forma de dependencia de desarrollo. Este paquete contiene las definiciones de tipos de Lambda.

```
npm install -D @types/aws-lambda
```

- Abra el directorio `lib`. Debería ver un archivo llamado `hello-world-stack.ts`. Cree dos nuevos archivos en este directorio: `hello-world.function.ts` y `hello-world.ts`.
- Abra `hello-world.function.ts` y agregue el siguiente código al archivo. Se trata del código de la función de Lambda.

### Note

La instrucción `import` importa las definiciones de tipo de [@types/aws-lambda](#). No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

- Abra `hello-world.ts` y agregue el siguiente código al archivo. Contiene la [NodejsFunction construcción](#), que crea la función Lambda, y la [LambdaRestApi construcción](#), que crea la API REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function');
    new LambdaRestApi(this, 'apigw', {
```

```

        handler: helloFunction,
    });
}
}

```

La construcción `NodejsFunction` presupone lo siguiente de manera predeterminada:

- El controlador de función se llama `handler`.
- El archivo `.ts` que contiene el código de la función (`hello-world.function.ts`) se encuentra en el mismo directorio que el archivo `.ts` que contiene la construcción (`hello-world.ts`). La construcción utiliza el ID de la construcción (“`hello-world`”) y el nombre del archivo del controlador de Lambda (“`function`”) para buscar el código de la función. Por ejemplo, si el código de la función se encuentra en un archivo llamado `hello-world.my-function.ts`, el archivo `hello-world.ts` debe hacer referencia al código de la función de este modo:

```
const helloFunction = new NodejsFunction(this, 'my-function');
```

Puede cambiar este comportamiento y configurar otros parámetros de `esbuild`. Para obtener más información, consulte [Configuración de esbuild](#) en la Referencia de la API de AWS CDK.

7. Abra `hello-world-stack.ts`. Este es el código que define la [pila de AWS CDK](#). Reemplace el código con lo siguiente:

```

import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
    constructor(scope: Construct, id: string, props?: StackProps) {
        super(scope, id, props);
        new HelloWorld(this, 'hello-world');
    }
}

```

8. el directorio `hello-world` que contiene su archivo `cdk.json`, implemente su aplicación.

```
cdk deploy
```

9. AWS CDK compila y empaqueta la función de Lambda mediante `esbuild` y, a continuación, implementa la función en el tiempo de ejecución de Lambda. El resultado muestra el punto de

conexión de la API REST. Abra el punto de conexión en un navegador para probar la función. Debería ver esta respuesta:

```
{"message": "hello world"}
```

Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

## Uso de AWS CLI y esbuild para implementar TypeScript código en Lambda

El siguiente ejemplo demuestra cómo transpilar e implementar TypeScript código en Lambda mediante esbuild y el AWS CLI. esbuild produce un archivo con todas las dependencias. JavaScript Este es el único archivo que hay que agregar al archivo .zip.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [Versión 2 de la AWS CLI](#)
- Node.js 18.x
- Un [rol de ejecución](#) para la función de Lambda.
- Para los usuarios de Windows, una utilidad de archivos zip como [7zip](#).

### Implementar una función de ejemplo

1. En su máquina local, cree un directorio de proyecto para su nueva función.
2. Cree un nuevo proyecto de Node.js con npm o un administrador de paquetes de su elección.

```
npm init
```

3. Agregue los paquetes [@types/aws-lambda](#) y [esbuild](#) en forma de dependencias de desarrollo. El paquete `@types/aws-lambda` contiene las definiciones de tipos de Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Cree un nuevo archivo con el nombre `index.ts`. Agregue el siguiente código al nuevo archivo. Se trata del código de la función de Lambda. La función devuelve el mensaje `hello world`. La función no crea ningún recurso de API Gateway.

**Note**

La instrucción `import` importa las definiciones de tipo de [@types/aws-lambda](#). No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. [Para obtener más información, consulte aws-lambda en el repositorio. DefinitelyTyped GitHub](#)

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

5. Agregue un script de compilación al archivo `package.json`. Esto configura `esbuild` para que cree automáticamente el paquete de implementación `.zip`. Para obtener más información, consulte [Build scripts](#) (Scripts de compilación) en la documentación de `esbuild`.

## Linux and MacOS

```
"scripts": {
  "prebuild": "rm -rf dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
  target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && zip -r index.zip index.js*"
},
```

## Windows

En este ejemplo, el comando `"postbuild"` usa la utilidad [7zip](#) para crear el archivo `.zip`. Utilice la utilidad `zip` de Windows que prefiera y modifique el comando según sea necesario.



```
"scripts": {
  "prebuild": "del /q dist",
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --target=es2020 --outfile=dist/index.js",
  "postbuild": "cd dist && 7z a -tzip index.zip index.js*"
},
```

6. Cree el paquete.

```
npm run build
```

7. Cree una función de Lambda mediante el paquete de implementación .zip. Reemplace el texto resaltado con el nombre de recurso de Amazon (ARN) del [rol de ejecución](#).

```
aws lambda create-function --function-name hello-world --runtime "nodejs18.x" --role arn:aws:iam::123456789012:role/lambda-ex --zip-file "fileb://dist/index.zip" --handler index.handler
```

8. [Ejecute un evento de prueba](#) para confirmar que la función devuelve la siguiente respuesta. Si desea invocar esta función mediante API Gateway, [cree y configure una API REST](#).

```
{
  "statusCode": 200,
  "body": "{\"message\": \"hello world\"}"
}
```

# Implementar código de TypeScript transpilado en Lambda con imágenes de contenedor

Puede implementar el código de TypeScript en una función de AWS Lambda en forma de [imagen de contenedor](#) de Node.js. AWS proporciona [imágenes base](#) para Node.js para ayudarle a crear la imagen de contenedor. Estas imágenes base están precargadas con un tiempo de ejecución de lenguaje y otros componentes necesarios para ejecutar la imagen en Lambda. AWS proporciona un archivo Dockerfile para cada una de las imágenes base para ayudar a crear su imagen contenedor.

Si utiliza una imagen base de la comunidad o de una empresa privada, debe [agregar el cliente de interfaz de tiempo de ejecución \(RIC\) de Node.js](#) a la imagen base para que sea compatible con Lambda.

Lambda proporciona un emulador de interfaz de tiempo de ejecución para realizar pruebas de manera local. Las imágenes base AWS para Node.js incluyen el emulador de interfaz de tiempo de ejecución. Si utiliza una imagen base alternativa, como una imagen de Alpine Linux o Debian, puede [compilar el emulador en su imagen](#) o [instalarlo en su equipo local](#).

## Uso de una imagen base de Node.js para compilar y empaquetar código de función de TypeScript

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#)
- Node.js 18.x

### Creación de una imagen a partir de una imagen base

Para crear una imagen a partir de una imagen base AWS para Lambda

1. En su máquina local, cree un directorio de proyecto para su nueva función.
2. Cree un nuevo proyecto de Node.js con npm o un administrador de paquetes de su elección.

```
npm init
```

3. Agregue los paquetes [@types/aws-lambda](#) y [esbuild](#) en forma de dependencias de desarrollo. El paquete `@types/aws-lambda` contiene las definiciones de tipos de Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Agregue un [script de compilación](#) al archivo `package.json`.

```
"scripts": {
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --target=es2020 --outfile=dist/index.js"
}
```

5. Cree un nuevo archivo denominado `index.ts`. Agregue el siguiente código de muestra al nuevo archivo. Se trata del código de la función de Lambda. La función devuelve el mensaje `hello world`.

#### Note

La instrucción `import` importa las definiciones de tipo de [@types/aws-lambda](#). No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio de GitHub DefinitelyTyped.

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

6. Cree un nuevo archivo `Dockerfile` con la siguiente configuración:
  - Establezca la propiedad `FROM` en el URI de la imagen base.

- Establezca el argumento `CMD` para especificar el controlador de función de Lambda.

## Example Dockerfile

El siguiente archivo Dockerfile utiliza una compilación de varias etapas. El primer paso transpila el código de TypeScript a JavaScript. El segundo paso produce una imagen de contenedor que contiene solo archivos de JavaScript y dependencias de producción.

```
FROM public.ecr.aws/lambda/nodejs:18 as builder
WORKDIR /usr/app
COPY package.json index.ts ./
RUN npm install
RUN npm run build

FROM public.ecr.aws/lambda/nodejs:18
WORKDIR ${LAMBDA_TASK_ROOT}
COPY --from=builder /usr/app/dist/* ./
CMD ["index.handler"]
```

7. Compile la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

## (Opcional) Prueba de la imagen de forma local

1. Inicie la imagen de Docker con el comando `docker run`. En este ejemplo, `docker-image` es el nombre de la imagen y `test` es la etiqueta.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

2. Desde una nueva ventana de terminal, publique un evento en el punto de conexión local.

#### Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

#### PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Obtenga el ID del contenedor.

```
docker ps
```

4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en



Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

# Uso del objeto de contexto Lambda para recuperar la información de la función de TypeScript

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución.

## Métodos de context

- `getRemainingTimeInMillis()`: devuelve el número de milisegundos que quedan antes del tiempo de espera de la ejecución.

## Propiedades de context

- `functionName`: el nombre de la función de Lambda.
- `functionVersion`: la [versión](#) de la función.
- `invokedFunctionArn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `memoryLimitInMB`: cantidad de memoria asignada a la función.
- `awsRequestId`: el identificador de la solicitud de invocación.
- `logGroupName`: grupo de registros de para la función.
- `logStreamName`: el flujo de registro de la instancia de la función.
- `identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
  - `cognitoIdentityId`: la identidad autenticada de Amazon Cognito.
  - `cognitoIdentityPoolId`: el grupo de identidad de Amazon Cognito que ha autorizado la invocación.
- `clientContext`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`

- `env.platform_version`
- `env.platform`
- `env.make`
- `env.model`
- `env.locale`
- Custom: valores personalizados que establece la aplicación del cliente.
- `callbackWaitsForEmptyEventLoop`: establézcalo en `false` para enviar la respuesta de forma inmediata cuando se ejecute la [devolución de llamada](#), en lugar de esperar a que el bucle de eventos de Node.js esté vacío. Si esto es `false`, los eventos pendientes siguen ejecutándose durante el siguiente periodo de invocación.

Puede usar el paquete de npm [@types/aws-lambda](#) para trabajar con el objeto de contexto.

Example archivo `index.ts`

La siguiente función de ejemplo registra información de contexto y devuelve la ubicación de los registros.

#### Note

Antes de utilizar este código en una función de Lambda, debe agregar el paquete [@types/aws-lambda](#) como dependencia de implementación. Este paquete contiene las definiciones de tipos de Lambda. Cuando `@types/aws-lambda` se encuentra instalado, la instrucción `import (import ... from 'aws-lambda')` importa las definiciones de tipo. No importa el paquete NPM de `aws-lambda`, que es una herramienta de terceros no relacionada. Para obtener más información, consulte [aws-lambda](#) en el repositorio de GitHub DefinitelyTyped.

```
import { Context } from 'aws-lambda';
export const lambdaHandler = async (event: string, context: Context): Promise<string>
=> {
  console.log('Remaining time: ', context.getRemainingTimeInMillis());
  console.log('Function name: ', context.functionName);
  return context.logStreamName;
};
```

# Registro y supervisión de las funciones de Lambda de TypeScript

AWS Lambda supervisa de forma automática funciones de Lambda y envía entradas de registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación y otros resultados del código de su función al flujo de registro. Para obtener más información acerca de Registros de CloudWatch, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Para generar registros desde el código de la función, puede utilizar los métodos del [objeto de la consola](#). Para un registro más detallado, puede utilizar cualquier biblioteca de registro que escriba en `stdout` o `stderr`.

## Secciones

- [Uso de herramientas y bibliotecas de registro](#)
- [Uso de Powertools para AWS Lambda \(TypeScript\) y AWS SAM para el registro estructurado](#)
- [Uso de Powertools para AWS Lambda \(TypeScript\) y el AWS CDK para el registro estructurado](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)

## Uso de herramientas y bibliotecas de registro

[Powertools para AWS Lambda \(TypeScript\)](#) es un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores. La [utilidad de logger](#) proporciona un logger optimizado para Lambda que incluye información adicional sobre el contexto de la función en todas las funciones con un resultado estructurado como JSON. Utilice esta utilidad para hacer lo siguiente:

- Capturar campos clave del contexto de Lambda, arranque en frío y resultados de registro de estructuras como JSON
- Registrar los eventos de invocación de Lambda cuando se le indique (desactivado de forma predeterminada)
- Imprimir todos los registros solo para un porcentaje de las invocaciones mediante el muestreo de registros (desactivado de forma predeterminada)
- Agregar claves adicionales al registro estructurado en cualquier momento

- Utilizar un formateador de registros personalizado (traiga su propio formateador) para generar registros en una estructura compatible con el RFC de registro de su organización

## Uso de Powertools para AWS Lambda (TypeScript) y AWS SAM para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación “Hola, mundo” de TypeScript de muestra con módulos de [Powertools para AWS Lambda \(TypeScript\)](#) integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Node.js 18.x o posterior
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

### Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación utilizando la plantilla de TypeScript de tipo Hola Mundo.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x
```


2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

 Note

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey=`HelloWorldApi`].OutputValue' --output text
```

6. Invoque el punto de conexión de la API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

7. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

El resultado del registro tendrá este aspecto:

```
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.552000  
START RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Version: $LATEST  
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.594000  
2022-08-31T09:33:10.557Z 70693159-7e94-4102-a2af-98a6343fb8fb  
INFO {"_aws":{"Timestamp":1661938390556,"CloudWatchMetrics":  
[{"Namespace":"sam-app","Dimensions":[["service"]],"Metrics":  
[{"Name":"ColdStart","Unit":"Count"}]}]}, "service":"helloWorld","ColdStart":1}  
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.595000  
2022-08-31T09:33:10.595Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO  
{"level":"INFO","message":"This is an INFO log - sending HTTP 200 - hello world  
response","service":"helloWorld","timestamp":"2022-08-31T09:33:10.594Z"}
```

```

2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.655000
2022-08-31T09:33:10.655Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
{"_aws":{"Timestamp":1661938390655,"CloudWatchMetrics":[{"Namespace":"sam-
app","Dimensions":[["service"]],"Metrics":[]}]}, "service":"helloWorld"}
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000 END
RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb
2023/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2022-08-31T09:33:10.754000
REPORT RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Duration: 201.55 ms Billed
Duration: 202 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 252.42
ms
XRAY TraceId: 1-630f2ad5-1de22b6d29a658a466e7ecf5 SegmentId: 567c116658fbf11a
Sampled: true

```

- Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Administración de retención de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros de forma indefinida, elimine el grupo de registros o configure un periodo de retención después del cual CloudWatch los eliminará de forma automática. Para configurar la retención de registros, agregue lo siguiente a la plantilla de AWS SAM:

```

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7

```

# Uso de Powertools para AWS Lambda (TypeScript) y el AWS CDK para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación “Hola, mundo” de TypeScript de muestra con módulos de [Powertools para AWS Lambda \(TypeScript\)](#) integrados mediante AWS CDK. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Node.js 18.x o posterior
- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world
cd hello-world
```

2. Inicialice la aplicación.

```
cdk init app --language typescript
```

3. Agregue el paquete [@types/aws-lambda](#) en forma de dependencia de desarrollo.

```
npm install -D @types/aws-lambda
```

4. Instale la [utilidad Logger](#) de Powertools.



```
npm install @aws-lambda-powertools/logger
```

- Abra el directorio lib. Debería ver un archivo llamado hello-world-stack.ts. Cree dos nuevos archivos en este directorio: hello-world.function.ts y hello-world.ts.
- Abra hello-world.function.ts y agregue el siguiente código al archivo. Se trata del código de la función de Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Logger } from '@aws-lambda-powertools/logger';
const logger = new Logger();

export const handler = async (event: APIGatewayEvent, context: Context):
Promise<APIGatewayProxyResult> => {
  logger.info('This is an INFO log - sending HTTP 200 - hello world response');
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

- Abra hello-world.ts y agregue el siguiente código al archivo. Contiene el [constructo NodejsFunction](#), que crea la función de Lambda, configura las variables de entorno para Powertools y establece la retención de registros en una semana. También incluye el [constructo LambdaRestApi](#), que crea la API de REST.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { RetentionDays } from 'aws-cdk-lib/aws-logs';
import { CfnOutput } from 'aws-cdk-lib';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        Powertools_SERVICE_NAME: 'helloWorld',
        LOG_LEVEL: 'INFO',
      },
    },
```

```

    logRetention: RetentionDays.ONE_WEEK,
  });
  const api = new LambdaRestApi(this, 'apigw', {
    handler: helloFunction,
  });
  new CfnOutput(this, 'apiUrl', {
    exportName: 'apiUrl',
    value: api.url,
  });
}
}

```

8. Abra `hello-world-stack.ts`. Este es el código que define la [pila de AWS CDK](#). Reemplace el código con lo siguiente:

```

import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}

```

9. Vuelva al directorio del proyecto.

```
cd hello-world
```

10. Implementación de la aplicación.

```
cdk deploy
```

11. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Invoque el punto de conexión de la API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

13. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```
sam logs --stack-name HelloWorldStack
```

El resultado del registro tendrá este aspecto:

```
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.047000
  START RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Version: $LATEST
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.050000 {
  "level": "INFO",
  "message": "This is an INFO log - sending HTTP 200 - hello world response",
  "service": "helloWorld",
  "timestamp": "2022-08-31T14:48:37.048Z",
  "xray_trace_id": "1-630f74c4-2b080cf77680a04f2362bcf2"
}
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000 END
  RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c
2023/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2022-08-31T14:48:37.082000
  REPORT RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Duration: 34.60 ms Billed
  Duration: 35 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 173.48
  ms
```

14. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
cdk destroy
```

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

### Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

# Seguimiento del código de TypeScript en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas tres bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK para Node.js](#): un SDK para generar y enviar datos de seguimiento a X-Ray.
- [Powertools para AWS Lambda \(TypeScript\)](#): un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de Powertools para AWS Lambda \(TypeScript\) y AWS SAM para el seguimiento](#)
- [Uso de Powertools para AWS Lambda \(TypeScript\) y el AWS CDK para el seguimiento](#)
- [Interpretación de un seguimiento de X-Ray](#)

# Uso de Powertools para AWS Lambda (TypeScript) y AWS SAM para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación “Hola, mundo” de TypeScript de muestra con módulos de [Powertools para AWS Lambda \(TypeScript\)](#) integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Node.js 18.x o posterior
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación utilizando la plantilla de TypeScript de tipo Hola Mundo.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs18.x --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima `Enter`.

**Note**

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

- Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

- invoque el punto de conexión de la API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

- Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
XRay Event [revision 1] at (2023-01-31T11:29:40.527000) with id  
(1-11a2222-111a222222cb33de3b95daf9) and duration (0.483s)  
- 0.425s - sam-app/Prod [HTTP: 200]  
- 0.422s - Lambda [HTTP: 200]  
- 0.406s - sam-app-HelloWorldFunction-XYZv11a1bcde [HTTP: 200]  
- 0.172s - sam-app-HelloWorldFunction-XYZv11a1bcde  
- 0.179s - Initialization  
- 0.112s - Invocation  
- 0.052s - ## app.lambdaHandler  
- 0.001s - ### MySubSegment  
- 0.059s - Overhead
```

- Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

## Uso de Powertools para AWS Lambda (TypeScript) y el AWS CDK para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación “Hola, mundo” de TypeScript de muestra con módulos de [Powertools para AWS Lambda \(TypeScript\)](#) integrados mediante AWS CDK. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Node.js 18.x o posterior
- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

### Implementar una aplicación de ejemplo de AWS Cloud Development Kit (AWS CDK)

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world
```



```
cd hello-world
```

## 2. Inicialice la aplicación.

```
cdk init app --language typescript
```

## 3. Agregue el paquete [@types/aws-lambda](#) en forma de dependencia de desarrollo.

```
npm install -D @types/aws-lambda
```

## 4. Instale la [utilidad Tracer](#) de Powertools.

```
npm install @aws-lambda-powertools/tracer
```

## 5. Abra el directorio lib. Debería ver un archivo llamado hello-world-stack.ts. Cree dos nuevos archivos en este directorio: hello-world.function.ts y hello-world.ts.

## 6. Abra hello-world.function.ts y agregue el siguiente código al archivo. Se trata del código de la función de Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Tracer } from '@aws-lambda-powertools/tracer';
const tracer = new Tracer();

export const handler = async (event: APIGatewayEvent, context: Context):
  Promise<APIGatewayProxyResult> => {
  // Get facade segment created by Lambda
  const segment = tracer.getSegment();

  // Create subsegment for the function and set it as active
  const handlerSegment = segment.addNewSubsegment(`## ${process.env._HANDLER}`);
  tracer.setSegment(handlerSegment);

  // Annotate the subsegment with the cold start and serviceName
  tracer.annotateColdStart();
  tracer.addServiceNameAnnotation();

  // Add annotation for the awsRequestId
  tracer.putAnnotation('awsRequestId', context.awsRequestId);
  // Create another subsegment and set it as active
  const subsegment = handlerSegment.addNewSubsegment('### MySubSegment');
  tracer.setSegment(subsegment);
  let response: APIGatewayProxyResult = {
```

```

    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  });
// Close subsegments (the Lambda one is closed automatically)
subsegment.close(); // (### MySubSegment)
handlerSegment.close(); // (## index.handler)

// Set the facade segment as active again (the one created by Lambda)
tracer.setSegment(segment);
return response;
};

```

7. Abra `hello-world.ts` y agregue el siguiente código al archivo. Contiene el [constructo `NodejsFunction`](#), que crea la función de Lambda, configura las variables de entorno para Powertools y establece la retención de registros en una semana. También incluye el [constructo `LambdaRestApi`](#), que crea la API de REST.

```

import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { CfnOutput } from 'aws-cdk-lib';
import { Tracing } from 'aws-cdk-lib/aws-lambda';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        POWERTOOLS_SERVICE_NAME: 'helloWorld',
      },
      tracing: Tracing.ACTIVE,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}

```

8. Abra `hello-world-stack.ts`. Este es el código que define la [pila de AWS CDK](#). Reemplace el código con lo siguiente:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Implemente la aplicación.

```
cd ..
cdk deploy
```

10. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Invoque el punto de conexión de la API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

12. Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id
(1-11a2222-111a222222cb33de3b95daf9) and duration (0.449s)
- 0.350s - HelloWorldStack-helloworldfunction111A2BCD-XYZv11a1bcde [HTTP: 200]
```

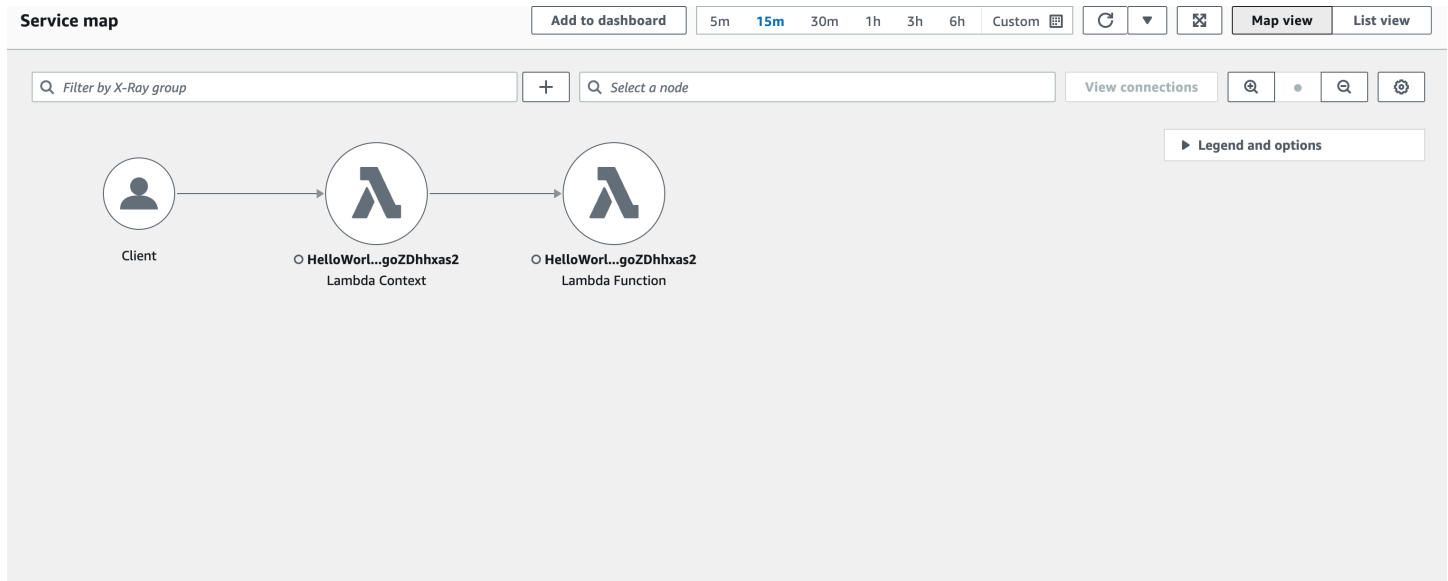
```
- 0.157s - HelloWorldStack-helloworldfunction111A2BCD-XYZv11a1bcde
- 0.169s - Initialization
- 0.058s - Invocation
  - 0.055s - ## index.handler
    - 0.000s - ### MySubSegment
- 0.099s - Overhead
```

13. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
cdk destroy
```

## Interpretación de un seguimiento de X-Ray

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [mapa de seguimiento de X-Ray](#) muestra información sobre la aplicación y todos sus componentes. En el siguiente ejemplo, se muestra un seguimiento de la aplicación de muestra:



## Creación de funciones de Lambda con Python

Puede ejecutar código Python en AWS Lambda. Lambda ofrece [tiempos de ejecución](#) para Python que ejecutan su código para procesar eventos. El código se ejecuta en un entorno que incluye el SDK for Python (Boto3), con credenciales de un rol de AWS Identity and Access Management (IAM) que usted administre. Para obtener más información sobre las versiones del SDK incluidas en los tiempos de ejecución de Python, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Lambda admite los siguientes tiempos de ejecución de Python.

### Python

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Python 3.12	python3.12	Amazon Linux 2023			
Python 3.11	python3.11	Amazon Linux 2			
Python 3.10	python3.10	Amazon Linux 2			
Python 3.9	python3.9	Amazon Linux 2			
Python 3.8	python3.8	Amazon Linux 2	14 de octubre de 2024	28 de febrero de 2025	31 de marzo de 2025

#### Note

La información de tiempo de ejecución en esta tabla se actualiza continuamente. Para obtener más información sobre el uso de los SDK de AWS en Lambda, consulte [Managing AWS SDKs in Lambda functions](#) en Serverless Land.

## Para crear una función Python

1. Abra la [consola de Lambda](#).
2. Elija Crear función.
3. Configure los siguientes ajustes:
  - En Nombre de la función: ingrese el nombre de la función.
  - Tiempo de ejecución: elija Python 3.12.
4. Elija Crear función.
5. Para configurar un evento de prueba, seleccione Prueba.
6. En Nombre del evento, escriba **test**.
7. Elija Guardar cambios.
8. Elija Test (Probar) para invocar la función.

La consola crea una función de Lambda con un único archivo de origen llamado `lambda_function`. Puede editar este archivo y agregar más archivos en el editor de código integrado. Para guardar los cambios, elija Guardar. A continuación, para ejecutar el código, elija Pruebas.

### Note

La consola de Lambda utiliza AWS Cloud9 para proporcionar un entorno de desarrollo integrado en el navegador. También puede utilizar AWS Cloud9 para desarrollar funciones de Lambda en su propio entorno. Para obtener más información, consulte [Trabajar con funciones de AWS Lambda utilizando AWS Toolkit](#) en la guía del usuario de AWS Cloud9.

### Note

Para comenzar con el desarrollo de aplicaciones en su entorno local, implemente una de las aplicaciones de ejemplo disponibles en el repositorio de GitHub de esta guía.

Aplicaciones de ejemplo de Lambda de ejemplo en Python

- [blank-ruby](#): una función de Python que muestra el uso del registro, las variables de entorno, el seguimiento de AWS X-Ray, las pruebas de unidad y el AWS SDK.

Su función de Lambda tiene un grupo de registros de Registros de CloudWatch. El tiempo de ejecución de la función envía detalles de cada invocación a Registros de CloudWatch. Se transmite cualquier [registro que su función genere](#) durante la invocación. Si su función devuelve un error, Lambda formatea el error y lo devuelve al invocador.

## Temas

- [Versiones del SDK incluidas en el tiempo de ejecución](#)
- [Formato de respuesta](#)
- [Cierre correcto de las extensiones](#)
- [Definir el controlador de funciones de Lambda en Python](#)
- [Uso de archivos .zip para funciones de Lambda en Python](#)
- [Implementar funciones de Python Lambda con imágenes de contenedor](#)
- [Uso de capas para funciones de Lambda en Python](#)
- [Uso del objeto de contexto Lambda para recuperar información de funciones de Python](#)
- [Registro y supervisión de las funciones de Lambda de Python](#)
- [Prueba de funciones de AWS Lambda en Python](#)
- [Instrumentación del código Python en AWS Lambda](#)

## Versiones del SDK incluidas en el tiempo de ejecución

La versión del AWS SDK incluida en el tiempo de ejecución de Python depende de la versión del tiempo de ejecución y de su Región de AWS. Para encontrar la versión del SDK incluida en el tiempo de ejecución que está utilizando, cree una función de Lambda con el código siguiente.

```
import boto3
import botocore

def lambda_handler(event, context):
    print(f'boto3 version: {boto3.__version__}')
    print(f'botocore version: {botocore.__version__}')
```

## Formato de respuesta

En los tiempos de ejecución de Python 3.12 y posteriores, las funciones devuelven caracteres Unicode como parte de su respuesta JSON. Los tiempos de ejecución de Python anteriores

devolvían secuencias con escape para caracteres Unicode en las respuestas. Por ejemplo, en Python 3.11, si devuelve una cadena Unicode como “こんにちは”, escapa a los caracteres Unicode y devuelve “\u3053\u3093\u306b\u3061\u306f”. El tiempo de ejecución de Python 3.12 devuelve el “こんにちは” original.

El uso de respuestas Unicode reduce el tamaño de las respuestas de Lambda, lo que facilita el ajuste de respuestas más grandes al tamaño máximo de carga útil de 6 MB para las funciones sincrónicas. En el ejemplo anterior, la versión con escape es de 32 bytes, en comparación con los 17 bytes de la cadena Unicode.

Cuando actualice a Python 3.12, es posible que tenga que ajustar el código para tener en cuenta el nuevo formato de respuesta. Si el llamador espera un escape de Unicode, debe agregar código a la función de retorno para escapar del Unicode manualmente, o ajustar el llamador para que pueda manejar la devolución de Unicode.

## Cierre correcto de las extensiones

Los tiempos de ejecución de Python 3.12 y posteriores ofrecen capacidades mejoradas de cierre correcto para funciones con [extensiones externas](#). Cuando Lambda cierra un entorno de ejecución, envía una señal SIGTERM al tiempo de ejecución y, a continuación, un evento SHUTDOWN a cada extensión externa registrada. Puede atrapar la señal SIGTERM en la función de Lambda y limpiar los recursos, como las conexiones a bases de datos, creados por la función.

Para obtener más información sobre el ciclo de vida del entorno de ejecución, consulte [Comprender el ciclo de vida del entorno de ejecución de Lambda](#). Para ver ejemplos de cómo usar un cierre elegante con extensiones, consulte el [repositorio GitHub de muestras de AWS](#).



# Definir el controlador de funciones de Lambda en Python

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

Puede usar la siguiente sintaxis general al crear un controlador de funciones en Python:

```
def handler_name(event, context):  
    ...  
    return some_value
```

## Temas

- [Denominación](#)
- [Funcionamiento](#)
- [Devolver un valor](#)
- [Ejemplos](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Python](#)

## Denominación

El nombre del controlador de funciones de Lambda especificado en el momento de la creación de una función de Lambda se deriva de:

- El nombre del archivo en el que se encuentra la función del controlador de Lambda.
- El nombre de la función del controlador de Python.

Un controlador de función puede ser cualquier nombre; sin embargo, el valor predeterminado en la consola de Lambda es `lambda_function.lambda_handler`. Este nombre del controlador de funciones refleja el nombre de la función (`lambda_handler`) y el archivo donde se almacena el código del controlador (`lambda_function.py`).

Si crea una función en la consola con un nombre de archivo o un nombre de controlador de funciones diferente, debe editar el nombre del controlador predeterminado.

Para cambiar el nombre de controlador de la función (consola)

1. Abra la página [Funciones](#) de la consola de Lambda y elija su función.
2. Elija la pestaña Código.
3. Desplácese hacia abajo hasta el panel Configuración del tiempo de ejecución y elija Editar.
4. En Controlador, ingrese el nuevo nombre del controlador de funciones.
5. Seleccione Guardar.

## Funcionamiento

Cuando Lambda invoca su controlador de función, [el tiempo de ejecución de Lambda](#) pasa dos argumentos al controlador de funciones:

- El primer argumento es el [objeto de evento](#). Un evento es un documento con formato JSON que contiene datos para que una función de Lambda los procese. El tiempo de [ejecución de Lambda](#) convierte el evento en un objeto y lo pasa al código de la función. Por lo general, es del tipo Python dict. También puede ser list, str, intfloat, o el tipo NoneType.

El objeto de evento contiene información del servicio de invocación. Cuando se invoca una función, se determina la estructura y el contenido del evento. Cuando un servicio AWS invoca su función, el servicio define la estructura del evento. Para obtener más información acerca de los eventos de los servicios de AWS, consulte [Invocar Lambda con eventos de otros servicios de AWS](#).

- El segundo argumento es el [objeto de contexto](#). Un objeto de contexto se pasa a su función de Lambda en tiempo de ejecución. Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de tiempo de ejecución.

## Devolver un valor

Si lo desea, el controlador puede devolver un valor. Lo que sucede con el valor devuelto depende del [tipo de invocación](#) y el [servicio](#) que invocó la función. Por ejemplo:

- Si utiliza el tipo de invocación RequestResponse, como [Invocación de una función de Lambda de forma sincrónica](#), AWS Lambda devuelve el resultado de la llamada a la función Python al cliente que invoca la función de Lambda (en la respuesta HTTP a la solicitud de invocación, serializado en JSON). Por ejemplo, la consola de AWS Lambda utiliza el tipo de invocación RequestResponse, por lo que al invocar la función utilizando la consola, esta mostrará el valor devuelto.

- Si el controlador devuelve objetos que no se pueden serializar con `json.dumps`, el tiempo de ejecución devuelve un error.
- Si el controlador devuelve `None`, al igual que hacen las funciones de Python sin una instrucción `return`, el runtime devuelve `null`.
- Si se utiliza el tipo de invocación Event (una [invocación asíncrona](#)), el valor se descarta.

### Note

En Python 3.9 y versiones posteriores, Lambda incluye el valor `requestId` de la invocación en la respuesta de error.

## Ejemplos

La siguiente sección muestra ejemplos de funciones de Python con las que puede usar Lambda. Si utiliza la consola de Lambda para crear la función, no es necesario adjuntar un [archivo .zip](#) para ejecutar las funciones de esta sección. Estas funciones usan bibliotecas estándar de Python que se incluyen con el tiempo de ejecución Lambda que seleccionó. Para obtener más información, consulte [???](#).

### Devolver un mensaje

En el siguiente ejemplo se muestra una función denominada `lambda_handler`. La función acepta la entrada del usuario de un nombre y apellido, y devuelve un mensaje que contiene datos del evento que recibió como entrada.

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

Puede utilizar los siguientes datos de evento para invocar la función:

```
{
  "first_name": "John",
  "last_name": "Smith"
}
```

La respuesta muestra los datos del evento pasados como entrada:

```
{
  "message": "Hello John Smith!"
}
```

## Analizar una respuesta

En el siguiente ejemplo se muestra una función denominada `lambda_handler`. La función utiliza datos de eventos pasados por Lambda en tiempo de ejecución. Analiza la [variable de entorno](#) en `AWS_REGION` devuelto en la respuesta JSON.

```
import os
import json

def lambda_handler(event, context):
    json_region = os.environ['AWS_REGION']
    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps({
            "Region ": json_region
        })
    }
```

Puede usar cualquier dato de evento para invocar la función:

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Los tiempos de ejecución de Lambda establecen varias variables de entorno durante la inicialización. Para obtener más información sobre las variables de entorno devueltas en la respuesta en tiempo de ejecución, consulte [Utilice variables de entorno Lambda para configurar valores en el código](#).

La función de este ejemplo depende de una respuesta exitosa (en `200`) de la API de Invoke. Para obtener más información sobre el estado de Invoke API, consulte Sintaxis de respuesta de [Invoke](#).

## Devolver un cálculo

En el siguiente ejemplo se muestra una función denominada `lambda_handler`. La función acepta la entrada del usuario y devuelve un cálculo al usuario. Para obtener más información sobre este ejemplo, consulte el [repositorio de GitHub `aws-doc-sdk-examples`](#).

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    ...
    result = None
    action = event.get('action')
    if action == 'increment':
        result = event.get('number', 0) + 1
        logger.info('Calculated result of %s', result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {'result': result}
    return response
```

Puede utilizar los siguientes datos de evento para invocar la función:

```
{
  "action": "increment",
  "number": 3
}
```

## Prácticas recomendadas de codificación para las funciones de Lambda en Python

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad. Por ejemplo, en Python, podría tener este aspecto:

```
def lambda_handler(event, context):
```

```
foo = event['foo']
bar = event['bar']

result = my_lambda_function(foo, bar)

def my_lambda_function(foo, bar):
    // MyLambdaFunction logic here
```

- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. Para los tiempos de ejecución de Node.js y Python, estos incluyen los SDK de AWS. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación.
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).

- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Uso de archivos .zip para funciones de Lambda en Python

El código de la función de AWS Lambda incluye un archivo .py que contiene el código del controlador de la función, junto con los paquetes y módulos adicionales de los que dependa el código. Para implementar este código de función en Lambda, se utiliza un paquete de despliegue. Este paquete puede ser un archivo de archivos .zip o una imagen de contenedor. Para obtener más información sobre el uso de imágenes de contenedores con Python, consulte [Implementar funciones de Lambda en Python con imágenes de contenedores](#).

Para crear un paquete de despliegue como un archivo de archivos .zip, puede emplear utilidad de archivado de archivos .zip incorporada de la herramienta de línea de comandos, o cualquier otra utilidad de archivos .zip, como [7zip](#). En los ejemplos que se muestran en las siguientes secciones se supone que está utilizando una herramienta zip de línea de comandos en un entorno Linux o macOS. Para utilizar los mismos comandos en Windows, puede [instalar el Subsistema de Windows para Linux](#) a fin de obtener una versión de Ubuntu y Bash integrada con Windows.

Tenga en cuenta que Lambda usa permisos de archivo de POSIX, por lo que puede necesitar [establecer permisos para la carpeta del paquete de despliegue](#) antes de crear el archivo de archivos .zip.

## Temas

- [Dependencias de tiempo de ejecución en Python](#)
- [Creación de un paquete de despliegue .zip sin dependencias](#)
- [Creación de un paquete de despliegue .zip con dependencias](#)
- [Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución](#)
- [Uso de carpetas \\_\\_pycache\\_\\_](#)
- [Creación de paquetes de despliegue .zip con bibliotecas nativas](#)
- [Creación y actualización de funciones de Lambda en Python mediante archivos .zip](#)

## Dependencias de tiempo de ejecución en Python

Para las funciones de Lambda que utilizan el tiempo de ejecución de Python, una dependencia puede ser cualquier paquete o módulo de Python. Cuando implemente la función mediante un archivo .zip, podrá agregar estas dependencias a su archivo .zip con el código de la función o utilizar una [capa de Lambda](#). Una capa es un archivo .zip independiente que puede contener código



adicional y otro contenido. Para obtener más información sobre el uso de las capas de Lambda en Python, consulte [the section called “Capas”](#).

Los tiempos de ejecución de Python para Lambda incluyen el AWS SDK for Python (Boto3) y sus dependencias. Lambda proporciona el SDK en el tiempo de ejecución para los escenarios de implementación en los que no puede agregar sus propias dependencias. Estos escenarios incluyen la creación de funciones en la consola mediante el editor de código integrado o el uso de funciones en línea en AWS Serverless Application Model (AWS SAM) o plantillas de AWS CloudFormation.

Lambda actualiza de manera periódica las bibliotecas del tiempo de ejecución de Python para incluir las actualizaciones y los parches de seguridad más recientes. Si la función usa la versión del SDK de Boto3 incluida en el tiempo de ejecución, pero el paquete de despliegue incluye las dependencias del SDK, esto puede provocar problemas de desalineación de versiones. Por ejemplo, el paquete de despliegue puede incluir la dependencia `urllib3` del SDK. Cuando Lambda actualiza el SDK en el tiempo de ejecución, los problemas de compatibilidad entre la nueva versión del tiempo de ejecución y la versión de `urllib3` del paquete de despliegue pueden provocar un error en la función.

#### Important

Para mantener un control total sobre las dependencias y evitar posibles problemas de desalineación de versiones, le recomendamos que agregue todas las dependencias de la función al paquete de despliegue, incluso si sus versiones están incluidas en el tiempo de ejecución de Lambda. Esto incluye el SDK de Boto3.

Para saber qué versión del SDK de Python (Boto3) se incluye en el tiempo de ejecución que está utilizando, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Según el [modelo de responsabilidad compartida de AWS](#), usted es responsable de la administración de cualquier dependencia en los paquetes de despliegue de sus funciones. Esto incluye la aplicación de actualizaciones y parches de seguridad. Para actualizar las dependencias del paquete de despliegue de la función, primero cree un nuevo archivo `.zip` y, a continuación, cárguelo en Lambda. Para obtener más información, consulte [Creación de un paquete de despliegue `.zip` con dependencias](#) y [Creación y actualización de funciones de Lambda en Python mediante archivos `.zip`](#).

## Creación de un paquete de despliegue `.zip` sin dependencias

Si el código de la función no tiene dependencias, el archivo `.zip` solo contiene el archivo `.py` con el código del controlador de la función. Utilice la utilidad `zip` que prefiera para crear un archivo `.zip` con

el archivo `.py` en la raíz. Si el archivo `.py` no está en la raíz del archivo `.zip`, Lambda no podrá ejecutar el código.

Para obtener información sobre cómo implementar un archivo `.zip` para crear una nueva función de Lambda o actualizar una existente, consulte [Creación y actualización de funciones de Lambda en Python mediante archivos .zip](#).

## Creación de un paquete de despliegue `.zip` con dependencias

Si el código de la función depende de paquetes o módulos adicionales, puede agregar estas dependencias al archivo `.zip` con el código de la función o utilizar una [capa de Lambda](#). En las instrucciones de esta sección, se muestra cómo incluir las dependencias en el paquete de despliegue `.zip`. Para que Lambda ejecute el código, el archivo `.py` que contiene el código del controlador y todas las dependencias de la función deben estar instalados en la raíz del archivo `.zip`.

Supongamos que el código de la función se guarda en un archivo llamado `lambda_function.py`. Los siguientes comandos de la CLI crean un archivo `.zip` llamado `my_deployment_package.zip` que contiene el código de la función de y sus dependencias. Puede instalar las dependencias directamente en una carpeta del directorio de su proyecto o utilizar un entorno virtual de Python.

Para crear el paquete de despliegue (directorio del proyecto)

1. Desplácese hasta el directorio del proyecto que contiene su archivo de código fuente `lambda_function.py`. En este ejemplo, el directorio se llama `my_function`.

```
cd my_function
```

2. Cree un nuevo directorio llamado “package” en el que instalará sus dependencias.

```
mkdir package
```

Tenga en cuenta que, en el caso de un paquete de despliegue `.zip`, Lambda espera que el código fuente y sus dependencias estén en la raíz del archivo `.zip`. Sin embargo, instalar dependencias directamente en el directorio del proyecto puede introducir una gran cantidad de archivos y carpetas nuevos, lo que dificulta la navegación por el IDE. Aquí se crea un directorio `package` independiente para mantener las dependencias separadas del código fuente.

3. Instale las dependencias en el directorio package. En el siguiente ejemplo, se instala el SDK de Boto3 desde el Índice de paquetes de Python mediante pip. Si el código de la función usa paquetes de Python que ha creado usted mismo, guárdelos en el directorio package.

```
pip install --target ./package boto3
```

4. Cree un archivo .zip con las bibliotecas instaladas en la raíz.

```
cd package
zip -r ../my_deployment_package.zip .
```

Esto genera un archivo `my_deployment_package.zip` en el directorio del proyecto.

5. Agregue el archivo `lambda_function.py` a la raíz del archivo .zip.

```
cd ..
zip my_deployment_package.zip lambda_function.py
```

El archivo .zip debe tener una estructura de directorios plana, con el código del controlador de la función y todas las carpetas de dependencias instalados en la raíz de la siguiente manera.

```
my_deployment_package.zip
|- bin
|  |-jp.py
|- boto3
|  |-compat.py
|  |-data
|  |-docs
...
|- lambda_function.py
```

Si el archivo .py que contiene el código del controlador de la función no está en la raíz del archivo .zip, Lambda no podrá ejecutar el código.

### Creación del paquete de despliegue (entorno virtual)

1. Cree y active un entorno virtual en el directorio del proyecto. En este ejemplo, el directorio del proyecto se llama `my_function`.

```
~$ cd my_function
```

```
~/my_function$ python3.12 -m venv my_virtual_env  
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Instale las bibliotecas necesarias con pip. En el siguiente ejemplo, se instala el SDK de Boto3.

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Use `pip show` para encontrar la ubicación en su entorno virtual donde pip ha instalado sus dependencias.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

La carpeta en la que pip instala sus bibliotecas puede llamarse `site-packages` o `dist-packages`. Esta carpeta puede estar ubicada en el directorio `lib/python3.x` o `lib64/python3.x` (donde `python3.x` representa la versión de Python que está utilizando).

4. Desactive el entorno virtual.

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Navegue al directorio que contiene las dependencias que instaló con pip y cree un archivo `.zip` en el directorio de su proyecto con las dependencias instaladas en la raíz. En este ejemplo, pip ha instalado sus dependencias en el directorio `my_virtual_env/lib/python3.12/site-packages`.

```
~/my_function$ cd my_virtual_env/lib/python3.12/site-packages  
~/my_function/my_virtual_env/lib/python3.12/site-packages$ zip -r ../../../../my_deployment_package.zip .
```

6. Navegue hasta la raíz del directorio de su proyecto donde se encuentra el archivo `.py` que contiene el código del controlador y agréguelo a la raíz del paquete `.zip`. En este ejemplo, el nombre del archivo de código de la función es `lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.12/site-packages$ cd ../../../../  
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

## Ruta de búsqueda de dependencias y bibliotecas incluidas en tiempo de ejecución

Cuando utilice una instrucción `import` en su código, el tiempo de ejecución de Python buscará en los directorios de su ruta de búsqueda hasta que encuentre el módulo o el paquete. De forma predeterminada, la primera ubicación que busca el tiempo de ejecución es el directorio en el que se descomprime y monta el paquete de despliegue `.zip` (`/var/task`). Si incluye una versión de una biblioteca incluida en el tiempo de ejecución de su paquete de implementación, la versión tendrá prioridad sobre la versión incluida en el tiempo de ejecución. Las dependencias del paquete de despliegue también tienen prioridad sobre las dependencias de las capas.

Cuando agregue una dependencia a una capa, Lambda la extraerá en `/opt/python/lib/python3.x/site-packages` (donde `python3.x` representa la versión del tiempo de ejecución que utiliza) o `/opt/python`. En la ruta de búsqueda, estos directorios tienen prioridad sobre los directorios que contienen las bibliotecas incluidas en el tiempo de ejecución y las bibliotecas instaladas con pip (`/var/runtime` y `/var/lang/lib/python3.x/site-packages`). Las bibliotecas de las capas de funciones tienen prioridad sobre las versiones incluidas en el tiempo de ejecución.

### Note

En la imagen base y el tiempo de ejecución administrado de Python 3.11, AWS SDK y sus dependencias se instalan en el directorio `/var/lang/lib/python3.11/site-packages`.

Para ver la ruta de búsqueda completa de la función de Lambda, agregue el siguiente fragmento de código.

```
import sys

search_path = sys.path
print(search_path)
```

### Note

Dado que las dependencias del paquete de despliegue o de las capas tienen prioridad sobre las bibliotecas incluidas en el tiempo de ejecución, esto puede provocar problemas de desalineación de versiones si incluye una dependencia del SDK, como `urllib3`, en su paquete

sin incluir también el SDK. Si implementa su propia versión de una dependencia de Boto3, también debe implementar Boto3 como una dependencia en su paquete de despliegue. Le recomendamos que empaquete todas las dependencias de la función, incluso si sus versiones están incluidas en el tiempo de ejecución.

También puede agregar dependencias en una carpeta independiente dentro del paquete .zip. Por ejemplo, puede agregar una versión del SDK de Boto3 a una carpeta del paquete .zip llamada `common`. Cuando descomprima y monte el paquete .zip, esta carpeta se colocará dentro del directorio `/var/task`. Para usar una dependencia de una carpeta del paquete de despliegue .zip en su código, utilice una instrucción `import from`. Por ejemplo, para usar una versión de Boto3 de una carpeta denominada `common` en el paquete .zip, utilice la siguiente instrucción.

```
from common import boto3
```

## Uso de carpetas `__pycache__`

Le recomendamos que no incluya carpetas `__pycache__` en el paquete de despliegue de la función. Es posible que el código de bytes de Python que se compila en un equipo de compilación con una arquitectura o un sistema operativo diferentes no sea compatible con el entorno de ejecución de Lambda.

## Creación de paquetes de despliegue .zip con bibliotecas nativas

Si su función solo usa paquetes y módulos puros de Python, puede utilizar el comando `pip install` para instalar las dependencias en cualquier equipo de compilación local y crear el archivo .zip. Muchas bibliotecas populares de Python, incluidas NumPy y Pandas, no son puras de Python y contienen código escrito en C o C++. Cuando agregue bibliotecas que contienen código C o C++ a su paquete de despliegue, deberá compilar el paquete correctamente para garantizar que sea compatible con el entorno de ejecución de Lambda.

La mayoría de los paquetes disponibles en el Índice de paquetes de Python ([PyPI](#)) están disponibles como “ruedas” (archivos .whl). Un archivo .whl es un tipo de archivo ZIP que contiene una distribución integrada con binarios precompilados para un sistema operativo y una arquitectura de conjunto de instrucciones en particular. Para que su paquete de despliegue sea compatible con Lambda, debe instalar la rueda para los sistemas operativos Linux y la arquitectura del conjunto de instrucciones de su función.

Es posible que algunos paquetes solo estén disponibles como distribuciones de código fuente. Para estos paquetes, debe compilar y crear los componentes de C o C++ usted mismo.

Para ver qué distribuciones están disponibles para el paquete que necesita, haga lo siguiente:

1. Busque el nombre del paquete en la [página principal del Índice de paquetes de Python](#).
2. Elija la versión del paquete que desea utilizar.
3. Seleccione Descargar archivos.

## Uso de distribuciones integradas (ruedas)

Para descargar una rueda que sea compatible con Lambda, utilice la opción `--platform` de pip.

Si la función de Lambda usa la arquitectura del conjunto de instrucciones x86\_64, ejecute el siguiente comando `pip install` para instalar una rueda compatible en su directorio `package`. Reemplace `--python 3.x` con la versión del tiempo de ejecución de Python que está utilizando.

```
pip install \  
--platform manylinux2014_x86_64 \  
--target=package \  
--implementation cp \  
--python-version 3.x \  
--only-binary=:all: --upgrade \  
<package_name>
```

Si la función usa la arquitectura del conjunto de instrucciones arm64, ejecute el siguiente comando. Reemplace `--python 3.x` con la versión del tiempo de ejecución de Python que está utilizando.

```
pip install \  
--platform manylinux2014_aarch64 \  
--target=package \  
--implementation cp \  
--python-version 3.x \  
--only-binary=:all: --upgrade \  
<package_name>
```

## Uso de distribuciones de código fuente

Si su paquete solo está disponible como distribución de código fuente, debe crear usted mismo las bibliotecas de C o C++. Para que su paquete sea compatible con el entorno de ejecución de Lambda,

debe crearlo en un entorno que utilice el mismo sistema operativo Amazon Linux 2. Para ello, cree su paquete en una instancia Linux de Amazon EC2.

Para obtener más información sobre cómo lanzar y conectarse a una instancia Linux de Amazon EC2, consulte [Tutorial: Introducción a las instancias Linux de Amazon EC2](#) en la Guía del usuario de Amazon EC2 para instancias Linux.

## Creación y actualización de funciones de Lambda en Python mediante archivos .zip

Una vez que haya creado su paquete de despliegue .zip, puede usarlo para crear una nueva función de Lambda o actualizar una existente. Puede implementar el paquete .zip a través de la consola, la AWS Command Line Interface y la API de Lambda. También puede crear y actualizar funciones de Lambda mediante AWS Serverless Application Model (AWS SAM) y AWS CloudFormation.

El tamaño máximo de un paquete de despliegue .zip para Lambda es de 250 MB (descomprimido). Tenga en cuenta que este límite se aplica al tamaño combinado de todos los archivos que cargue, incluidas las capas de Lambda.

El tiempo de ejecución de Lambda necesita permiso para leer los archivos del paquete de implementación. En la notación octal de permisos de Linux, Lambda necesita 644 permisos para archivos no ejecutables (rw-r--r--) y 755 permisos (rwxr-xr-x) para directorios y archivos ejecutables.

En Linux y macOS, utilice el comando `chmod` para cambiar los permisos de los archivos y directorios del paquete de implementación. Por ejemplo, para brindarle a un archivo ejecutable los permisos correctos, ejecute el siguiente comando.

```
chmod 755 <filepath>
```

Para cambiar los permisos de los archivos en Windows, consulte [Set, View, Change, or Remove Permissions on an Object](#) en la documentación de Microsoft Windows.


## Creación y actualización de funciones con archivos .zip mediante la consola

Para crear una nueva función, primero debe crearla en la consola y, a continuación, cargar el archivo .zip. Para actualizar una función existente, abra la página de la función correspondiente y, a continuación, siga el mismo procedimiento para agregar el archivo .zip actualizado.

Si el archivo .zip tiene un tamaño inferior a 50 MB, puede crear o actualizar una función al cargarlo directamente desde su equipo local. Para archivos .zip de más de 50 MB, primero debe cargar su



paquete en un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS Management Console, consulte [Introducción a Amazon S3](#). Para cargar archivos mediante la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

 Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

Para crear una nueva función (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija Crear función.
2. Elija Crear desde cero.
3. En Información básica, haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función.
  - b. En Tiempo de ejecución, seleccione el tiempo de ejecución que desea utilizar.
  - c. (Opcional) Para Arquitectura, elija la arquitectura del conjunto de instrucciones para su función. La arquitectura predeterminada es x86\_64. Asegúrese de que el paquete de despliegue .zip para su función sea compatible con la arquitectura del conjunto de instrucciones que seleccione.
4. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o utilizar uno existente.
5. Elija Crear función. Lambda crea una función básica “Hola, mundo” mediante el tiempo de ejecución elegido.

Para cargar un archivo .zip desde su equipo local (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar el archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija un archivo .zip.

5. Para cargar el archivo .zip, haga lo siguiente:
  - a. Seleccione Cargar y, a continuación, seleccione su archivo .zip en el selector de archivos.
  - b. Elija Abrir.
  - c. Seleccione Guardar.

#### Carga de un archivo .zip desde un bucket de Amazon S3 (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar un nuevo archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija la ubicación de Amazon S3.
5. Pegue la URL del enlace de Amazon S3 de su archivo .zip y seleccione Guardar.

#### Actualización de las funciones del archivo .zip mediante el editor de código de la consola

Para algunas funciones con paquetes de despliegue en formato .zip, puede utilizar el editor de código integrado en la consola de Lambda para actualizar el código de la función de forma directa. Para utilizar esta característica, la función debe cumplir los siguientes criterios:

- La función debe utilizar uno de los tiempos de ejecución del lenguaje interpretado (Python, Node.js o Ruby)
- El paquete de despliegue de la función debe tener un tamaño inferior a 3 MB.

El código de función de las funciones con paquetes de despliegue de imágenes de contenedores no se puede editar directamente en la consola.

Para actualizar el código de función mediante el editor de código de la consola

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, seleccione su archivo de código fuente y edítelo en el editor de código integrado.

4. Cuando haya terminado de editar el código, seleccione Implementar para guardar los cambios y actualizar la función.

## Creación y actualización de funciones con archivos .zip mediante la AWS CLI

Puede utilizar la [AWS CLI](#) para crear una nueva función o actualizar una existente con un archivo .zip. Utilice los comandos [create-function](#) y [update-function-code](#) para implementar su paquete .zip. Si el archivo .zip tiene un tamaño inferior a 50 MB, puede cargarlo desde una ubicación de archivo en su equipo de compilación local. Para archivos más grandes, debe cargar su paquete .zip desde un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

### Note

Si carga su archivo .zip desde un bucket de Amazon S3 con la AWS CLI, el bucket debe estar ubicado en la misma Región de AWS que su función.

Para crear una nueva función mediante un archivo .zip con la AWS CLI, debe especificar lo siguiente:

- El nombre de la función (`--function-name`).
- El tiempo de ejecución de la función (`--runtime`).
- El nombre de recurso de Amazon (ARN) del [rol de ejecución](#) de la función (`--role`).
- El nombre del método de controlador en el código de la función (`--handler`).

También debe especificar la ubicación del archivo .zip. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda create-function --function-name myFunction \  
--runtime python3.12 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice la opción `--code`, como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `S3ObjectVersion` para los objetos con versiones.

```
aws lambda create-function --function-name myFunction \  
--runtime python3.12 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Para actualizar una función existente mediante la CLI, especifique el nombre de la función mediante el parámetro `--function-name`. También debe especificar la ubicación del archivo `.zip` que desea utilizar para actualizar el código de la función. Si el archivo `.zip` se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo `.zip` en un bucket de Amazon S3, utilice las opciones `--s3-bucket` y `--s3-key` tal como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `--s3-object-version` para los objetos con versiones.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creación y actualización de funciones con archivos `.zip` mediante la API de Lambda

Para crear y actualizar funciones con un archivo de archivos `.zip`, utilice las siguientes operaciones de la API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creación y actualización de funciones con archivos `.zip` mediante AWS SAM

AWS Serverless Application Model (AWS SAM) es un conjunto de herramientas que ayuda a agilizar el proceso de creación y ejecución de aplicaciones sin servidor en AWS. Defina los recursos de su aplicación en una plantilla YAML o JSON y utilice la interfaz de la línea de comandos de AWS SAM (AWS SAM CLI) para crear, empaquetar e implementar sus aplicaciones. Al crear una función de Lambda a partir de una plantilla de AWS SAM, AWS SAM crea automáticamente un paquete de

despliegue .zip o una imagen de contenedor con el código de la función y las dependencias que especifique. Para obtener más información sobre el uso de AWS SAM para crear e implementar funciones de Lambda, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

También puede utilizar AWS SAM para crear una función de Lambda con un archivo de archivos .zip existente. Para crear una función de Lambda mediante AWS SAM, puede guardar el archivo .zip en un bucket de Amazon S3 o en una carpeta local de su equipo de compilación. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS SAM, el recurso `AWS::Serverless::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `CodeUri`: se establece como el URI de Amazon S3, la ruta a la carpeta local o el objeto [FunctionCode](#) del código de la función.
- `Runtime`: se establece como el entorno de ejecución elegido

Con AWS SAM, si su archivo .zip tiene más de 50 MB, no es necesario cargarlo primero en un bucket de Amazon S3. AWS SAM puede cargar paquetes .zip hasta el tamaño máximo permitido de 250 MB (descomprimidos) desde una ubicación de su equipo de compilación local.

Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS SAM.

## Creación y actualización de funciones con archivos .zip mediante AWS CloudFormation

Puede utilizar AWS CloudFormation para crear una función de Lambda con un archivo de archivos .zip. Para crear una función de Lambda a partir de un archivo .zip, primero debe cargar el archivo a un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

Para los tiempos de ejecución de Node.js y Python, también puede proporcionar código fuente en línea en la plantilla de AWS CloudFormation. A continuación, AWS CloudFormation crea un archivo .zip que contiene el código de la función creada.

## Uso de un archivo .zip existente

En la plantilla de AWS CloudFormation, el recurso `AWS::Lambda::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `Code`: ingrese el nombre del bucket de Amazon S3 y el nombre del archivo .zip en los campos `S3Bucket` y `S3Key`.
- `Runtime`: se establece como el entorno de ejecución elegido

## Creación de un archivo .zip a partir de código en línea

Puede declarar funciones simples escritas en Python o Node.js en línea en una plantilla de AWS CloudFormation. Como el código está incrustado en YAML o JSON, no puede agregar ninguna dependencia externa a su paquete de despliegue. Esto significa que la función debe usar la versión del AWS SDK que se incluye en el tiempo de ejecución. Los requisitos de la plantilla, como tener que usar caracteres de escape para ciertos caracteres, también dificultan el uso de las características de comprobación de sintaxis y finalización de código del IDE. Esto significa que la plantilla puede requerir pruebas adicionales. Debido a estas limitaciones, declarar funciones en línea es más adecuado para códigos muy simples que no cambian con frecuencia.

Para crear un archivo .zip a partir de código en línea para los tiempos de ejecución de Node.js y Python, establezca las siguientes propiedades en el recurso `AWS::Lambda::Function` de la plantilla:

- `PackageType`: se establece como `Zip`.
- `Code`: se ingresa el código de la función en el campo `ZipFile`.
- `Runtime`: se establece como el tiempo de ejecución elegido.

El archivo .zip que genera AWS CloudFormation no puede superar los 4 MB. Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS CloudFormation, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

# Implementar funciones de Python Lambda con imágenes de contenedor

Hay tres formas de crear una imagen de contenedor para una función de Lambda en Python:

- [Uso de una imagen base de AWS para Python](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Python](#) en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Python](#) en la imagen.

 Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.

## Temas

- [AWS imágenes base para Python](#)
- [Uso de una imagen base de AWS para Python](#)
- [Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución](#)

## AWS imágenes base para Python

AWS proporciona las siguientes imágenes base para Python:

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
3.12	Python 3.12	Amazon Linux 2023	<a href="#">Dockerfile para Python 3.12 en GitHub</a>	
3.11	Python 3.11	Amazon Linux 2	<a href="#">Dockerfile para Python 3.11 en GitHub</a>	
3.10	Python 3.10	Amazon Linux 2	<a href="#">Dockerfile para Python 3.10 en GitHub</a>	
3.9	Python 3.9	Amazon Linux 2	<a href="#">Dockerfile para Python 3.9 en GitHub</a>	
3.8	Python 3.8	Amazon Linux 2	<a href="#">Dockerfile para Python 3.8 en GitHub</a>	14 de octubre de 2024

Repositorio de Amazon ECR: [gallery.ecr.aws/lambda/python](https://gallery.ecr.aws/lambda/python)

Las imágenes base de Python 3.12 y versiones posteriores se basan en la [imagen de contenedor mínima de Amazon Linux 2023](#). Las imágenes de Python 3.8-3.11 se basan en la imagen de Amazon Linux 2. (). Las imágenes basadas en AL2023 ofrecen varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como `glibc`.

Las imágenes basadas en AL2023 utilizan `microdnf` (enlazado de forma simbólica como `dnf`) como administrador de paquetes en lugar de `yum`, que es el administrador de paquetes



predeterminado en Amazon Linux 2. `microdnf` es una implementación independiente de `dnf`. Para obtener una lista de los paquetes que se incluyen en las imágenes basadas en AL2023, consulte las columnas de Minimal Container de [Comparing packages installed on Amazon Linux 2023 Container Images](#). Para obtener más información sobre las diferencias entre AL2023 y Amazon Linux 2, consulte [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) en el Blog de informática de AWS.

#### Note

Para ejecutar imágenes basadas en AL2023 de forma local, incluso con AWS Serverless Application Model (AWS SAM), debe usar Docker en la versión 20.10.10 o posterior.

## Ruta de búsqueda de dependencias en las imágenes base

Cuando utilice una instrucción `import` en su código, el tiempo de ejecución de Python buscará en los directorios de su ruta de búsqueda hasta que encuentre el módulo o el paquete. De forma predeterminada, el tiempo de ejecución busca en el directorio `{LAMBDA_TASK_ROOT}` primero. Si incluye una versión de una biblioteca incluida en el tiempo de ejecución de su imagen, esta versión tendrá prioridad sobre la versión incluida en el tiempo de ejecución.

Los demás pasos de la ruta de búsqueda dependen de la versión de la imagen base de Lambda para Python que utilice:

- Python 3.11 y posteriores: las bibliotecas incluidas en el tiempo de ejecución y las bibliotecas instaladas con `pip` se instalan en el directorio `/var/lang/lib/python3.11/site-packages`. Este directorio tiene prioridad sobre `/var/runtime` en la ruta de búsqueda. Puede anular el SDK mediante `pip` para instalar una versión más reciente. Puede usar `pip` para comprobar que el SDK incluido en el tiempo de ejecución y sus dependencias son compatibles con los paquetes que instale.
- Python 3.8-3.10: las bibliotecas incluidas en el tiempo de ejecución se instalan en el directorio `/var/runtime`. Las bibliotecas instaladas con `pip` se instalan en el directorio `/var/lang/lib/python3.x/site-packages`. El directorio `/var/runtime` tiene prioridad sobre `/var/lang/lib/python3.x/site-packages` en la ruta de búsqueda.

Para ver la ruta de búsqueda completa de la función de Lambda, agregue el siguiente fragmento de código.

```
import sys

search_path = sys.path
print(search_path)
```

## Uso de una imagen base de AWS para Python

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#) (versión mínima 20.10.10 para imágenes base de Python 3.12 y versiones posteriores)
- Python

### Creación de una imagen a partir de una imagen base

Para crear una imagen de contenedor a partir de una imagen base de AWS para Python

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
cd example
```

2. Cree un nuevo archivo denominado `lambda_function.py`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

#### Example Función de Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. Cree un nuevo archivo denominado `requirements.txt`. Si utiliza el código de función de ejemplo del paso anterior, puede dejar el archivo vacío porque no hay dependencias. De lo contrario, enumere cada biblioteca requerida. Por ejemplo, este es el aspecto que `requirements.txt` debería tener si la función usa AWS SDK for Python (Boto3):

### Example requirements.txt

```
boto3
```

4. Cree un nuevo archivo Dockerfile con la siguiente configuración:
  - Establezca la propiedad FROM en el [URI de la imagen base](#).
  - Utilice el comando COPY para copiar el código de la función y las dependencias del tiempo de ejecución a `{LAMBDA_TASK_ROOT}`, una [variable de entorno definido de Lambda](#).
  - Establezca el argumento CMD para el controlador de la función de Lambda.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/python:3.12

# Copy requirements.txt
COPY requirements.txt ${LAMBDA_TASK_ROOT}

# Install the specified packages
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
of the Dockerfile)
CMD [ "lambda_function.handler" ]
```

5. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función

de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

(Opcional) Prueba de la imagen de forma local

1. Inicie la imagen de Docker con el comando `docker run`. En este ejemplo, `docker-image` es el nombre de la imagen y `test` es la etiqueta.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

2. Desde una nueva ventana de terminal, publique un evento en el punto de conexión local.

#### Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

#### PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

3. Obtenga el ID del contenedor.

```
docker ps
```

4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",
```

```
"statusCode": 200
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

## Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución

Si usa una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir el cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución extiende el [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de la función.

Instale el [cliente de interfaz de tiempo de ejecución para Python](#) mediante el administrador de paquetes pip:

```
pip install awslambdaric
```

También puede descargar el [cliente de interfaz de tiempo de ejecución de Python](#) desde GitHub.

En el siguiente ejemplo, se muestra cómo crear una imagen de contenedor para Python mediante una imagen base que no es de AWS. El Dockerfile de ejemplo usa una imagen base oficial de Python. El Dockerfile incluye el cliente de interfaz de tiempo de ejecución para Python.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)



- [Docker](#)
- Python

## Creación de imágenes a partir de una imagen base alternativa

Para crear una imagen de contenedor a partir de una imagen base que no es de AWS

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
cd example
```

2. Cree un nuevo archivo denominado `lambda_function.py`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

### Example Función de Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. Cree un nuevo archivo denominado `requirements.txt`. Si utiliza el código de función de ejemplo del paso anterior, puede dejar el archivo vacío porque no hay dependencias. De lo contrario, enumere cada biblioteca requerida. Por ejemplo, este es el aspecto que `requirements.txt` debería tener si la función usa AWS SDK for Python (Boto3):

### Example requirements.txt

```
boto3
```

4. Cree un nuevo Dockerfile. El siguiente Dockerfile usa una imagen base oficial de Python en lugar de una [imagen base de AWS](#). El Dockerfile incluye el [cliente de interfaz de tiempo de ejecución](#), que hace que la imagen sea compatible con Lambda. El siguiente Dockerfile de ejemplo utiliza una [compilación de varias etapas](#).
  - Establezca la propiedad FROM como la imagen base.
  - Configure ENTRYPOINT como el módulo que desea que el contenedor de Docker ejecute cuando se inicie. En este caso, el módulo es el cliente de interfaz de tiempo de ejecución.
  - Establezca el CMD como el controlador de la función de Lambda.

## Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM python:3.12 as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

# Install the function's dependencies
RUN pip install \
    --target ${FUNCTION_DIR} \
    awslambdagic

# Use a slim version of the base Python image to reduce the final image size
FROM python:3.12-slim

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdagic" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "lambda_function.handler" ]
```

5. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

**Note**

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

**(Opcional) Prueba de la imagen de forma local**

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. Puede [crear el emulador en su imagen](#) o usar el procedimiento siguiente para instalarlo en su equipo local.

Para instalar y ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Desde el directorio del proyecto, ejecute el siguiente comando para descargar el emulador de interfaz de tiempo de ejecución (arquitectura x86-64) de GitHub e instalarlo en su equipo local.

**Linux/macOS**

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Para instalar el emulador arm64, reemplace la URL del repositorio de GitHub en el comando anterior por lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

**PowerShell**

```
$dirPath = "$HOME\.aws-lambda-rie"  
if (-not (Test-Path $dirPath)) {  
  New-Item -Path $dirPath -ItemType Directory  
}
```

```
$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Para instalar el emulador arm64, reemplace el `$downloadLink` con lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:

- `docker-image` es el nombre de la imagen y `test` es la etiqueta.
- `/usr/local/bin/python -m awslambdaric lambda_function.handler` es el ENTRYPOINT seguido del CMD de su Dockerfile.

### Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  /usr/local/bin/python -m awslambdaric lambda_function.handler
```

### PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
/usr/local/bin/python -m awslambdaric lambda_function.handler
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

**Note**

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

3. Publique un evento en el punto de conexión local.

**Linux/macOS**

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{"payload": "hello world!"}'
```

**PowerShell**

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload": "hello world!"}' -ContentType "application/json"
```

4. Obtenga el ID del contenedor.

```
docker ps
```

5. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
```

```

    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.

7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.



---

Para obtener un ejemplo de cómo crear una imagen de Python a partir de una imagen base Alpine, consulte [Soporte de imágenes de contenedor para Lambda](#) en el Blog AWS.

# Uso de capas para funciones de Lambda en Python

Una [capa de Lambda](#) es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración. La creación de una capa implica tres pasos generales:

1. Empaquete el contenido de su capa. Esto significa crear un archivo de archivo. zip que contenga las dependencias que desea usar en sus funciones.
2. Cree la capa en Lambda.
3. Agregue la capa a sus funciones.

Este tema contiene los pasos y las instrucciones sobre cómo empaquetar y crear correctamente una capa de Lambda en Python con dependencias de bibliotecas externas.

## Temas

- [Requisitos previos](#)
- [Compatibilidad de capas de Python con Amazon Linux](#)
- [Rutas de capa para tiempos de ejecución de Python](#)
- [Empaquetado del contenido de la capa](#)
- [Creación de la capa](#)
- [Agregue la capa a la función](#)
- [Trabajo con ruedas de distribuciones manylinux](#)

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [Python 3.11](#) y el instalador de paquetes [pip](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

A lo largo de este tema, haremos referencia a la aplicación de muestra [layer-python](#) en el repositorio de GitHub awsdocs. Esta aplicación contiene scripts que descargan las dependencias y generan las capas. La aplicación también contiene las funciones correspondientes que utilizan las dependencias de las capas. Tras crear una capa, puede implementar e invocar la función

correspondiente para comprobar que todo funciona correctamente. Como utiliza el tiempo de ejecución de Python 3.11 para las funciones, las capas también deben ser compatibles con Python 3.11.

En la aplicación de muestra `layer-python`, hay dos ejemplos:

- El primer ejemplo implica empaquetar la biblioteca [requests](#) en una capa de Lambda. El directorio `layer/` contiene los scripts para generar la capa. El directorio `function/` contiene una función de ejemplo que ayuda a comprobar el funcionamiento de la capa. La mayor parte de este tutorial explica cómo crear y empaquetar esta capa.
- El segundo ejemplo implica empaquetar la biblioteca [numpy](#) en una capa de Lambda. El directorio `layer-numpy/` contiene los scripts para generar la capa. El directorio `function-numpy/` contiene una función de ejemplo que ayuda a comprobar el funcionamiento de la capa. Para ver un ejemplo de cómo crear y empaquetar esta capa, consulte [the section called “Trabajo con ruedas de distribuciones manylinux”](#).

## Compatibilidad de capas de Python con Amazon Linux

El primer paso para crear una capa consiste en agrupar todo el contenido de la capa en un archivo `.zip`. Dado que las funciones de Lambda se ejecutan en [Amazon Linux](#), el contenido de la capa debe poder compilarse y crearse en un entorno de Linux.

En Python, la mayoría de los paquetes están disponibles como [ruedas](#) (archivos `.whl`) además de la distribución de código fuente. Cada rueda es un tipo de distribución creada que admite una combinación específica de versiones de Python, sistemas operativos y conjuntos de instrucciones de máquina.

Las ruedas son útiles para garantizar que la capa sea compatible con Amazon Linux. Cuando descargue sus dependencias, descargue la rueda universal si es posible. (De forma predeterminada, `pip` instala la rueda universal si hay alguna disponible). La rueda universal contiene `any` como etiqueta de plataforma, lo que indica que es compatible con todas las plataformas, incluida Amazon Linux.

En el siguiente ejemplo, empaqueta la biblioteca `requests` en una capa de Lambda. La biblioteca `requests` es un ejemplo de un paquete que está disponible como rueda universal.

No todos los paquetes de Python se distribuyen como ruedas universales. Por ejemplo, [numpy](#) tiene varias distribuciones de ruedas, cada una de las cuales admite un conjunto diferente de plataformas.

Para estos paquetes, descargue la distribución `manylinux` para garantizar la compatibilidad con Amazon Linux. Para obtener instrucciones detalladas sobre cómo empaquetar dichas capas, consulte [the section called “Trabajo con ruedas de distribuciones `manylinux`”](#).

En raras ocasiones, es posible que un paquete de Python no esté disponible como rueda. Si solo existe la [distribución de origen](#) (`sdist`), se recomienda instalar y empaquetar las dependencias en un entorno [Docker](#) basado en la [imagen del contenedor base de Amazon Linux 2023](#). También recomendamos este enfoque si desea incluir sus propias bibliotecas personalizadas escritas en otros lenguajes, como C/C++. Este enfoque imita el entorno de ejecución de Lambda en Docker y garantiza que las dependencias de paquetes que no son de Python sean compatibles con Amazon Linux.

## Rutas de capa para tiempos de ejecución de Python

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio `/opt` de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable `PATH` ya incluye rutas de carpeta específicas en el directorio `/opt`. Para garantizar que la variable `PATH` recoja el contenido de la capa, el archivo `.zip` de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

- `python`
- `python/lib/python3.x/site-packages`

Por ejemplo, el archivo `.zip` de capa resultante que cree en este tutorial tiene la siguiente estructura de directorios:

```
layer_content.zip
# python
  # lib
    # python3.11
      # site-packages
        # requests
        # <other_dependencies> (i.e. dependencies of the requests package)
        # ...
```

La biblioteca [requests](#) está ubicada correctamente en el directorio `python/lib/python3.11/site-packages`. Esto garantiza que Lambda pueda localizar la biblioteca durante las invocaciones de funciones.

## Empaquetado del contenido de la capa

En este ejemplo, empaqueta la biblioteca `requests` de Python en un archivo `.zip` de capa. Siga los pasos que se indican a continuación para instalar y empaquetar el contenido de la capa.

Para instalar y empaquetar el contenido de la capa

1. Clone el [repositorio de `aws-lambda-developer-guide` de GitHub](https://github.com/awsdocs/aws-lambda-developer-guide), que contiene el código de muestra que necesita en el directorio `sample-apps/layer-python`.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Navegue hasta el directorio `layer` de la aplicación de ejemplo `layer-python`. Este directorio contiene los scripts que usa para crear y empaquetar la capa correctamente.

```
cd aws-lambda-developer-guide/sample-apps/layer-python/layer
```

3. Examine el archivo [requirements.txt](#). Este archivo define las dependencias que desea incluir en la capa, es decir, la biblioteca `requests`. Puede actualizar este archivo para incluir cualquier dependencia que desee incluir en su propia capa.

Example requirements.txt

```
requests==2.31.0
```

4. Asegúrese de tener los permisos para ejecutar ambos scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Ejecute el script [1-install.sh](#) mediante el siguiente comando:

```
./1-install.sh
```

Este script utiliza `venv` para crear un entorno virtual de Python llamado `create_layer`. A continuación, instala todas las dependencias necesarias en el directorio `create_layer/lib/python3.11/site-packages`.

Example 1-install.sh

```
python3.11 -m venv create_layer
```

```
source create_layer/bin/activate
pip install -r requirements.txt
```

6. Ejecute el script [2-package.sh](#) mediante el siguiente comando:

```
./2-package.sh
```

Este script copia el contenido del directorio `create_layer/lib` a un nuevo directorio denominado `python`. Luego, comprime el contenido del directorio `python` en un archivo llamado `layer_content.zip`. Este es el archivo `.zip` para su capa. Puede descomprimir el archivo y comprobar que contiene la estructura de archivos correcta, como se muestra en la sección [the section called “Rutas de capa para tiempos de ejecución de Python”](#).

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

## Creación de la capa

En esta sección, seleccione el archivo `layer_content.zip` que generó en la sección anterior y cárguelo como una capa de Lambda. Puede cargar una capa mediante la AWS Management Console o la API de Lambda en la AWS Command Line Interface (AWS CLI). Al cargar el archivo `.zip` de la capa, en el siguiente comando de AWS CLI [PublishLayerVersion](#), especifique `python3.11` como tiempo de ejecución compatible y `arm64` como arquitectura compatible.

```
aws lambda publish-layer-version --layer-name python-requests-layer \
  --zip-file fileb://layer_content.zip \
  --compatible-runtimes python3.11 \
  --compatible-architectures "arm64"
```

En la respuesta, anote el `LayerVersionArn`, que tiene este aspecto `arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1`. Necesitará este nombre de recurso de Amazon (ARN) en el siguiente paso de este tutorial cuando agregue la capa a la función.

## Agregue la capa a la función

En esta sección, despliega una función de Lambda de ejemplo que utiliza la biblioteca `requests` en su código de función y, a continuación, adjunta la capa. Para implementar la función, necesita un [the section called “Rol de ejecución \(permisos para que las funciones accedan a otros recursos\)”](#). Si no dispone de un rol de ejecución existente, siga los pasos de la sección desplegable. De no ser así, vaya a la siguiente sección para implementar la función.

(Opcional) Cree un rol de ejecución

Para crear un rol de ejecución

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Trusted entity (Entidad de confianza).–Lambda:.
  - Permisos: `AWSLambdaBasicExecutionRole`.
  - Nombre de rol: **lambda-role**.

La política `AWSLambdaBasicExecutionRole` tiene permisos que la función necesita para escribir registros a Registros de CloudWatch.

Para implementar la función de Lambda

1. Vaya al directorio `function/`. Si se encuentra actualmente en el directorio `layer/`, ejecute el siguiente comando:

```
cd ../function
```

2. Revise el [código de la función](#). La función importa la biblioteca `requests`, realiza una simple solicitud HTTP GET y a continuación devuelve el código del estado y el cuerpo.

```
import requests

def lambda_handler(event, context):
    print(f"Version of requests library: {requests.__version__}")
    request = requests.get('https://api.github.com/')
```

```
return {
    'statusCode': request.status_code,
    'body': request.text
}
```

3. Cree un archivo .zip del paquete de implementación utilizando el siguiente comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Implemente la función. En el siguiente comando AWS CLI, sustituya el parámetro `--role` por el ARN del rol de ejecución:

```
aws lambda create-function --function-name python_function_with_layer \
    --runtime python3.11 \
    --architectures "arm64" \
    --handler lambda_function.lambda_handler \
    --role arn:aws:iam::123456789012:role/lambda-role \
    --zip-file fileb://my_deployment_package.zip
```

(Opcional) Invocar su función sin adjuntar una capa

En este punto, si lo desea, puede intentar invocar su función antes de adjuntar la capa. Si lo intenta, aparecerá un error de importación porque la función no puede hacer referencia al paquete `requests`. Utilice el comando AWS CLI para invocar la función:

```
aws lambda invoke --function-name python_function_with_layer \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{
  "StatusCode": 200,
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST"
}
```

Para ver el error específico, abra el archivo de salida `response.json`. Debería ver un `ImportModuleError` con el siguiente mensaje de error:



```
"errorMessage": "Unable to import module 'lambda_function': No module named 'requests'"
```

A continuación, adjunte la capa a la función. En el siguiente comando AWS CLI, sustituya el parámetro `--layers` por el ARN de la versión de capa que indicó anteriormente:

```
aws lambda update-function-configuration --function-name python_function_with_layer \  
--cli-binary-format raw-in-base64-out \  
--layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

Finalmente, intente invocar su función usando el siguiente comando AWS CLI:

```
aws lambda invoke --function-name python_function_with_layer \  
--cli-binary-format raw-in-base64-out \  
--payload '{"key": "value"}' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"  
}
```

El archivo de salida `response.json` contiene detalles sobre la respuesta.

(Opcional) Eliminar sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

Para eliminar la capa de Lambda

1. Abra la página de [Capas](#) de la consola de Lambda.
2. Seleccione la capa que ha creado.
3. Elija Eliminar, luego, vuelva a elegir Eliminar.

Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.

2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete(Eliminar).

## Trabajo con ruedas de distribuciones **manylinux**

A veces, un paquete que desee incluir como dependencia no tendrá una rueda universal (específicamente, no tendrá una etiqueta de plataforma any). En este caso, descargue la rueda `manylinux` compatible. Esto garantiza que las bibliotecas de capas sean compatibles con Amazon Linux.

[numpy](#) es un paquete que no tiene una rueda universal. Si desea incluir el paquete `numpy` en la capa, puede completar los siguientes pasos de ejemplo para instalar y empaquetar la capa correctamente.

Para instalar y empaquetar el contenido de la capa

1. Clone el [repositorio de aws-lambda-developer-guide de GitHub](#), que contiene el código de muestra que necesita en el directorio `sample-apps/layer-python`.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Navegue hasta el directorio `layer-numpy` de la aplicación de ejemplo `layer-python`. Este directorio contiene los scripts que usa para crear y empaquetar la capa correctamente.

```
cd aws-lambda-developer-guide/sample-apps/layer-python/layer-numpy
```

3. Examine el archivo [requirements.txt](#). Este archivo define las dependencias que desea incluir en la capa, es decir, la biblioteca `numpy`. Aquí, especifica la URL de la rueda de distribución `manylinux` que es compatible con Python 3.11, Amazon Linux y el conjunto de instrucciones `x86_64`:

Example requirements.txt

```
https://files.pythonhosted.org/packages/3a/d0/edc009c27b406c4f9cbc79274d6e46d634d139075492ad055e3d68445925/numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

4. Asegúrese de tener los permisos para ejecutar ambos scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Ejecute el script [1-install.sh](#) mediante el siguiente comando:

```
./1-install.sh
```

Este script utiliza venv para crear un entorno virtual de Python llamado `create_layer`. A continuación, instala todas las dependencias necesarias en el directorio `create_layer/lib/python3.11/site-packages`. El comando `pip` es diferente en este caso, ya que debe especificar la etiqueta `--platform` como `manylinux2014_x86_64`. Esto le indica a `pip` que instale la rueda `manylinux` correcta, incluso si su máquina local usa macOS o Windows.

Example 1-install.sh

```
python3.11 -m venv create_layer
source create_layer/bin/activate
pip install -r requirements.txt --platform=manylinux2014_x86_64 --only-binary=:all:
--target ./create_layer/lib/python3.11/site-packages
```

6. Ejecute el script [2-package.sh](#) mediante el siguiente comando:

```
./2-package.sh
```

Este script copia el contenido del directorio `create_layer/lib` a un nuevo directorio denominado `python`. Luego, comprime el contenido del directorio `python` en un archivo llamado `layer_content.zip`. Este es el archivo `.zip` para su capa. Puede descomprimir el archivo y comprobar que contiene la estructura de archivos correcta, como se muestra en la sección [the section called “Rutas de capa para tiempos de ejecución de Python”](#).

Example 2-package.sh

```
mkdir python
cp -r create_layer/lib python/
zip -r layer_content.zip python
```

Para cargar esta capa a Lambda, utilice el siguiente comando de la AWS CLI [PublishLayerVersion](#):

```
aws lambda publish-layer-version --layer-name python-numpy-layer \
```

```
--zip-file fileb://layer_content.zip \  
--compatible-runtimes python3.11 \  
--compatible-architectures "x86_64"
```

En la respuesta, anote el `LayerVersionArn`, que tiene este aspecto `arn:aws:lambda:us-east-1:123456789012:layer:python-numpy-layer:1`. Para comprobar que la capa funciona según lo previsto, despliegue la función de Lambda en el directorio `function-numpy`.

Para implementar la función de Lambda

1. Vaya al directorio `function-numpy/`. Si se encuentra actualmente en el directorio `layer-numpy/`, ejecute el siguiente comando:

```
cd ../function-numpy
```

2. Revise el [código de la función](#). La función importa la biblioteca `numpy`, crea una matriz `numpy` simple y luego devuelve un código de estado y un cuerpo ficticios.

```
import json  
import numpy as np  
  
def lambda_handler(event, context):  
  
    x = np.arange(15, dtype=np.int64).reshape(3, 5)  
    print(x)  
  
    return {  
        'statusCode': 200,  
        'body': json.dumps('Hello from Lambda!')  
    }
```

3. Cree un archivo `.zip` del paquete de implementación utilizando el siguiente comando:

```
zip my_deployment_package.zip lambda_function.py
```

4. Implemente la función. En el siguiente comando AWS CLI, sustituya el parámetro `--role` por el ARN del rol de ejecución:

```
aws lambda create-function --function-name python_function_with_numpy \  
--runtime python3.11 \  
--handler lambda_function.lambda_handler \  
--role arn:aws:iam::123456789012:role/lambda-role \  

```

```
--zip-file fileb://my_deployment_package.zip
```

(Opcional) Invocar su función sin adjuntar una capa

En este punto, si lo desea, puede intentar invocar su función antes de adjuntar la capa. Si lo intenta, aparecerá un error de importación porque la función no puede hacer referencia al paquete `numpy`. Utilice el comando AWS CLI para invocar la función:

```
aws lambda invoke --function-name python_function_with_numpy \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "key": "value" }' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "StatusCode": 200,  
  "FunctionError": "Unhandled",  
  "ExecutedVersion": "$LATEST"  
}
```

Para ver el error específico, abra el archivo de salida `response.json`. Debería ver un `ImportModuleError` con el siguiente mensaje de error:

```
"errorMessage": "Unable to import module 'lambda_function': No module named 'numpy'"
```

A continuación, adjunte la capa a la función. En el siguiente comando AWS CLI, sustituya el parámetro `--layers` por el ARN de su versión de capa:

```
aws lambda update-function-configuration --function-name python_function_with_numpy \  
  --cli-binary-format raw-in-base64-out \  
  --layers "arn:aws:lambda:us-east-1:123456789012:layer:python-requests-layer:1"
```

Finalmente, intente invocar su función usando el siguiente comando AWS CLI:

```
aws lambda invoke --function-name python_function_with_numpy \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "key": "value" }' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"  
}
```

Puede examinar los registros de funciones para comprobar que el código imprime la matriz numpy de forma estándar.

# Uso del objeto de contexto Lambda para recuperar información de funciones de Python

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución. Para obtener más información sobre cómo se pasa el objeto de contexto al controlador de funciones, consulte [Definir el controlador de funciones de Lambda en Python](#).

## Métodos de context

- `get_remaining_time_in_millis`: devuelve el número de milisegundos que quedan antes del tiempo de espera de la ejecución.

## Propiedades de context

- `function_name`: el nombre de la función de Lambda.
- `function_version`: la [versión](#) de la función.
- `invoked_function_arn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `memory_limit_in_mb`: cantidad de memoria asignada a la función.
- `aws_request_id`: el identificador de la solicitud de invocación.
- `log_group_name`: grupo de registros de para la función.
- `log_stream_name`: el flujo de registro de la instancia de la función.
- `identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
  - `cognito_identity_id`: la identidad autenticada de Amazon Cognito.
  - `cognito_identity_pool_id`: el grupo de identidad de Amazon Cognito que ha autorizado la invocación.
- `client_context`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`

- `client.app_package_name`
- `custom`: un elemento `dict` de valores personalizados establecidos por la aplicación cliente para móviles.
- `env`: un elemento `dict` de información de entorno proporcionado por el AWS SDK.

En el siguiente ejemplo se muestra una función de controlador que registra información de context.

#### Example handler.py

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:", context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining in
    get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", context.get_remaining_time_in_millis())
```

Además de las opciones que se enumeran anteriormente, también puede utilizar el SDK de X-Ray AWS para [Instrumentación del código Python en AWS Lambda](#) a fin de identificar las rutas de código críticas, rastrear su rendimiento y capturar los datos para su análisis.



# Registro y supervisión de las funciones de Lambda de Python

AWS Lambda supervisa de forma automática funciones de Lambda y envía entradas de registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación y otros resultados del código de su función al flujo de registro. Para obtener más información acerca de Registros de CloudWatch, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Para generar registros a partir del código de función, puede utilizar el módulo [logging](#) integrado. Para entradas más detalladas, puede utilizar cualquier biblioteca de registro que escriba en `stdout` o `stderr`.

## Impresión en el registro

Para enviar el resultado básico a los registros, puede utilizar un método `print` en su función. En el siguiente ejemplo se registran los valores de grupo de registro y flujo de Registros de CloudWatch y el objeto de evento.

Tenga en cuenta que, si su función genera registros mediante instrucciones de Python `print`, Lambda solo puede enviar las salidas de registro a Registros de CloudWatch en formato de texto sin formato. Para capturar registros en JSON estructurado, debe utilizar una biblioteca de registros compatible. Para obtener más información, consulte [the section called “Uso de los controles de registro avanzados de Lambda con Python”](#).

Example `lambda_function.py`

```
import os
def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    print(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    print('## EVENT')
    print(event)
```

Example salida del registro

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
/aws/lambda/my-function
```

```
2023/08/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
  Duration: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1  SegmentId: 07f5xmpl2d1f6f85
  Sampled: true
```

El tiempo de ejecución de Python registra las líneas START, END y REPORT de cada invocación. La línea REPORT incluye los siguientes datos:

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.
- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- TraceId de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

## Uso de una biblioteca de registro

Para obtener registros más detallados, utilice el módulo de [registro](#) en la biblioteca estándar o en cualquier biblioteca de registro de terceros que escriba en `stdout` o `stderr`.

Para los tiempos de ejecución de Python compatibles, puede elegir si los registros creados con el módulo `logging` estándar se capturan en texto sin formato o JSON. Para obtener más información, consulte [the section called “Uso de los controles de registro avanzados de Lambda con Python”](#).

Actualmente, el formato de registro predeterminado para todos los tiempos de ejecución de Python es texto sin formato. El siguiente ejemplo muestra cómo las salidas de registro creados con el módulo `logging` estándar se capturan en texto sin formato en Registros de CloudWatch.

```
import os
import logging
logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    logger.info(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    logger.info('## EVENT')
    logger.info(event)
```

La salida de `logger` incluye el nivel de registro, la marca de tiempo y el ID de la solicitud.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO] 2023-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ##
ENVIRONMENT VARIABLES
[INFO] 2023-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 /aws/
lambda/my-function
[INFO] 2023-08-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 2023/01/31/
[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d
[INFO] 2023-08-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ## EVENT
[INFO] 2023-08-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 {'key':
'value'}
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Duration: 2.75 ms Billed
Duration: 3 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861
Sampled: true
```

### Note

Cuando el formato de registro de la función está establecido en texto sin formato, la configuración de nivel de registro predeterminada para los tiempos de ejecución de Python es `WARN`. Esto significa que Lambda envía a Registros de CloudWatch solo las salidas de registro de nivel `WARN` e inferiores. Para cambiar el nivel de registro predeterminado, utilice el método `setLevel()` de `logging` de Python como se muestra en este código de ejemplo. Si establece el formato de registro de la función en JSON, le recomendamos que configure el nivel de registro de la función mediante los controles de registro avanzados de Lambda y no

configure el nivel de registro en el código. Para obtener más información, consulte [the section called “Uso del filtrado a nivel de registro con Python”](#)

## Uso de los controles de registro avanzados de Lambda con Python

Para tener más control sobre cómo se registran, procesan y consumen los registros de sus funciones, puede configurar las siguientes opciones de registro para los tiempos de ejecución de Lambda Python admitidos:

- Formato de registro: seleccione entre texto sin formato y el formato JSON estructurado para los registros de su función
- Nivel de registro: para los registros en formato JSON, elija el nivel de detalle de los registros que Lambda envía a Amazon CloudWatch, como ERROR, DEBUG o INFO
- Grupo de registro: elija el grupo de registro de CloudWatch al que su función envía los registros

Para obtener más información sobre estas opciones de registro e instrucciones sobre cómo configurar la función para utilizarlas, consulte [the section called “Configuración de registros de funciones”](#).

Para utilizar las opciones de formato de registro y nivel de registro con las funciones de Lambda de Python, consulte las instrucciones en las siguientes secciones.

### Uso de registros JSON estructurados con Python

Si selecciona JSON para el formato de registro de su función, Lambda enviará los registros de salida utilizando la biblioteca de registros estándar de Python a CloudWatch como JSON estructurado. Cada objeto de registro JSON contiene, por lo menos, cuatro pares clave-valor con las siguientes claves:

- "timestamp": la hora en que se generó el mensaje de registro
- "level": el nivel de registro asignado al mensaje
- "message": el contenido del mensaje de registro
- "requestId": el ID de solicitud único para la invocación de la función

La biblioteca de logging de Python también puede agregar pares clave-valor adicionales, como "logger", a este objeto JSON.

Los ejemplos de las siguientes secciones muestran cómo las salidas de registro generadas con la biblioteca de logging de Python se capturan en Registros de CloudWatch cuando se configura el formato de registro de la función como JSON.

Tenga en cuenta que si utiliza el método `print` para generar salidas de registro básicos como se describe en [the section called “Impresión en el registro”](#), Lambda capturará estas salidas como texto sin formato, incluso si configura el formato de registro de la función como JSON.

### Salidas de registro JSON estándar mediante la biblioteca de registro de Python

El siguiente ejemplo de fragmento de código y de salidas de registro muestra cómo las salidas de registro estándar generadas con la biblioteca de logging de Python se capturan en Registros de CloudWatch cuando se configura el formato de registro de la función como JSON.

#### Example Código de registro de Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

#### Example Entrada de registro JSON

```
{
  "timestamp": "2023-10-27T19:17:45.586Z",
  "level": "INFO",
  "message": "Inside the handler function",
  "logger": "root",
  "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

### Registro de parámetros adicionales en JSON

Cuando el formato de registros de la función se establece en JSON, también puede registrar parámetros adicionales con la biblioteca logging de Python estándar utilizando la clave `extra` para pasar a un diccionario de Python a la salida del registro.

#### Example Código de registro de Python

```
import logging
```

```
def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
        extra={"a": "b", "b": [3]},
    )
```

### Example Entrada de registro JSON

```
{
  "timestamp": "2023-11-02T15:26:28Z",
  "level": "INFO",
  "message": "extra parameters example",
  "logger": "root",
  "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",
  "a": "b",
  "b": [
    3
  ]
}
```

### Registro de excepciones en JSON

El siguiente fragmento de código muestra cómo se capturan las excepciones de Python en la salida del registro de su función cuando configura el formato de registro como JSON. Tenga en cuenta que a las salidas de registro generadas mediante `logging.exception` se les asigna el nivel de registro `ERROR`.

### Example Código de registro de Python

```
import logging

def lambda_handler(event, context):
    try:
        raise Exception("exception")
    except:
        logging.exception("msg")
```

### Example Entrada de registro JSON

```
{
  "timestamp": "2023-11-02T16:18:57Z",
```

```

"level": "ERROR",
"message": "msg",
"logger": "root",
"stackTrace": [
  " File \"/var/task/lambda_function.py\", line 15, in lambda_handler\n      raise
Exception(\\"exception\\")\n"
],
"errorType": "Exception",
"errorMessage": "exception",
"requestId": "3f9d155c-0f09-46b7-bdf1-e91dab220855",
"location": "/var/task/lambda_function.py:lambda_handler:17"
}

```

## Registros estructurados en JSON con otras herramientas de registro

Si su código ya usa otra biblioteca de registro, como Powertools para AWS Lambda, para producir registros estructurados en JSON, no necesita realizar ningún cambio. AWS Lambda no codifica dos veces ningún registro que ya esté codificado en JSON. Incluso si configura su función para utilizar el formato de registro JSON, las salidas del registro aparecen en CloudWatch en la estructura JSON que defina.

El siguiente ejemplo muestra cómo se capturan en Registros de CloudWatch las salidas de registro generadas con el AWS Lambda paquete Powertools. El formato de la salida de este registro es el mismo independientemente de si la configuración de registro de la función está establecida en JSON o TEXT. Para obtener más información acerca del uso de Powertools para AWS Lambda, consulte [the section called “Uso de Powertools para AWS Lambda \(Python\) y AWS SAM para el registro estructurado”](#) y [the section called “Uso de Powertools para AWS Lambda \(Python\) y AWS CDK para el registro estructurado”](#)

### Example Fragmento de código de registro de Python (usando Powertools para AWS Lambda)

```

from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")

```

### Example Entrada de registro JSON (usando Powertools para AWS Lambda)

```
{
```

```
"level": "INFO",
"location": "lambda_handler:7",
"message": "Inside the handler function",
"timestamp": "2023-10-31 22:38:21,010+0000",
"service": "service_undefined",
"xray_trace_id": "1-654181dc-65c15d6b0fecbdd1531ecb30"
}
```

## Uso del filtrado a nivel de registro con Python

Al configurar el filtrado a nivel de registro, puede elegir enviar solo los registros de un nivel de registro específico o inferior a Registros de CloudWatch. Para obtener información sobre cómo configurar el filtrado a nivel de registro para su función, consulte [the section called “Filtrado a nivel de registro”](#).

Para que AWS Lambda filtre los registros de las aplicaciones según su nivel de registro, la función debe usar registros con formato JSON. Puede lograr esto de dos maneras:

- Cree salidas de registro con la biblioteca `logging` estándar de Python y configure su función para que utilice el formato de registro JSON. A continuación, AWS Lambda filtra las salidas del registro utilizando el par clave-valor “nivel” del objeto JSON descrito en [the section called “Uso de registros JSON estructurados con Python”](#). Para obtener información sobre cómo configurar el formato de registro de la función, consulte [the section called “Configuración de registros de funciones”](#).
- Utilice otra biblioteca o método de registro para crear registros estructurados en JSON en su código que incluyan un par clave-valor “nivel” que defina el nivel de la salida del registro. Por ejemplo, puede utilizar Powertools para AWS Lambda con el objetivo de generar salidas de registros JSON estructurados a partir de su código.

También puede utilizar una instrucción de impresión para generar un objeto JSON que contenga un identificador de nivel de registro. La siguiente instrucción de impresión produce una salida con formato JSON en la que el nivel de registro se establece en INFO. AWS Lambda enviará el objeto JSON a Registros de CloudWatch si el nivel de registro de la función está establecido en INFO, DEBUG o TRACE.

```
print({'msg':"My log message", "level":"info"})
```

Para que Lambda filtre los registros de la función, también debe incluir un par clave-valor “timestamp” en la salida del registro JSON. La hora debe especificarse con un formato de marca



de tiempo [RFC 3339](#) válido. Si no proporciona una marca de tiempo válida, Lambda asignará al registro el nivel INFO y agregará una marca de tiempo por usted.

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

### Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros con AWS CLI

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

### Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgt0GNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

### Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
```

```
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad base64 está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar base64 -D.

### Example get-logs.sh script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza sed para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando get-log-events.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como get-logs.sh.

La opción cli-binary-format es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute aws configure set cli-binary-format raw-in-base64-out. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

### Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
}
```

```
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"  
}
```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

## Uso de otras herramientas y bibliotecas de registro

[Powertools para AWS Lambda \(Python\)](#) es un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores. La [utilidad de logger](#) proporciona un logger optimizado para Lambda que incluye información adicional sobre el contexto de la función en todas las funciones con un resultado estructurado como JSON. Utilice esta utilidad para hacer lo siguiente:

- Capturar campos clave del contexto de Lambda, arranque en frío y resultados de registro de estructuras como JSON
- Registrar los eventos de invocación de Lambda cuando se le indique (desactivado de forma predeterminada)
- Imprimir todos los registros solo para un porcentaje de las invocaciones mediante el muestreo de registros (desactivado de forma predeterminada)
- Agregar claves adicionales al registro estructurado en cualquier momento
- Utilizar un formateador de registros personalizado (traiga su propio formateador) para generar registros en una estructura compatible con el RFC de registro de su organización

## Uso de Powertools para AWS Lambda (Python) y AWS SAM para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Python con módulos de [Powertools para Python](#) integrados mediante el AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Python 3.9
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla “Hola, mundo” de Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type Zip --runtime python3.9 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

### Note

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque el punto de conexión de la API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

7. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

El resultado del registro tendrá este aspecto:

```
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
2023-02-03T14:59:50.371000 INIT_START Runtime Version:
python:3.9.v16 Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app>HelloWorldFunction-YBg8yfYt0c9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
>HelloWorldFunction-YBg8yfYt0c9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
```

```

        [
            "function_name",
            "service"
        ]
    ],
    "Metrics": [
        {
            "Name": "ColdStart",
            "Unit": "Count"
        }
    ]
}
]
},
"function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
"service": "PowertoolsHelloWorld",
"ColdStart": [
    1.0
]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  }
}
],
"service": "PowertoolsHelloWorld",
>HelloWorldInvocations": [
    1.0
]
]

```



```

}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
Billed Duration: 17 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
Sampled: true

```

- Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Administración de retención de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros de forma indefinida, elimine el grupo de registros o configure un periodo de retención después del cual CloudWatch los eliminará de forma automática. Para configurar la retención de registros, agregue lo siguiente a la plantilla de AWS SAM:

```

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7

```

## Uso de Powertools para AWS Lambda (Python) y AWS CDK para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Python con módulos integrados de [Powertools para AWS Lambda \(Python\)](#) mediante AWS CDK. Esta aplicación implementa un backend de API básico y

utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje hello world.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Python 3.9
- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world  
cd hello-world
```

2. Inicialice la aplicación.

```
cdk init app --language python
```

3. Instale las dependencias de Python.

```
pip install -r requirements.txt
```

4. Cree un directorio lambda\_function en la carpeta raíz.

```
mkdir lambda_function  
cd lambda_function
```

5. Cree un archivo app.py y agregue el siguiente código al archivo. Se trata del código de la función de Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
```

```
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
                      value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
# ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

6. Abra el directorio `hello_world`. Debería ver un archivo llamado `hello_world_stack.py`.

```
cd ..
cd hello_world
```

7. Abra `hello_world_stack.py` y agregue el siguiente código al archivo. Esto contiene el [constructor de Lambda](#), el cual crea la función de Lambda, configura variables de entorno para Powertools y

establece la retención de registros en una semana, y el [constructor ApiGatewayV1](#), el cual crea la API de REST.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
            # constructor is in Alpha, o we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
            aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
            docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_9,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
```

```

        "LOG_LEVEL": "INFO"
    }
)

apigw = apigwv1.RestApi(self, "PowertoolsAPI",
    deploy_options=apigwv1.StageOptions(stage_name="dev"))

hello_api = apigw.root.add_resource("hello")
hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
    proxy=True))

CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")

```

## 8. Implementación de la aplicación.

```

cd ..
cdk deploy

```

## 9. Obtenga la URL de la aplicación implementada:

```

aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text

```

## 10. Invoque el punto de conexión de la API:

```

curl GET <URL_FROM_PREVIOUS_STEP>

```

Si se realiza de forma correcta, verá el siguiente resultado:

```

{"message":"hello world"}

```

## 11. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```

sam logs --stack-name HelloWorldStack

```

El resultado del registro tendrá este aspecto:

```

2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
2023-02-03T14:59:50.371000 INIT_START Runtime Version:

```

```

python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.112000
START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2023-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYt0c9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYt0c9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}

```

```

]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000 END
RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2023/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2023-02-03T14:59:51.128000
REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be   Duration: 16.33 ms
Billed Duration: 17 ms   Memory Size: 128 MB   Max Memory Used: 64 MB   Init
Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299   SegmentId: 3c5d18d735a1ced0
Sampled: true

```

12. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
cdk destroy
```

# Prueba de funciones de AWS Lambda en Python

## Note

Consulte el capítulo [Prueba de funciones](#) para obtener una introducción completa a las técnicas y prácticas recomendadas para probar soluciones sin servidor.

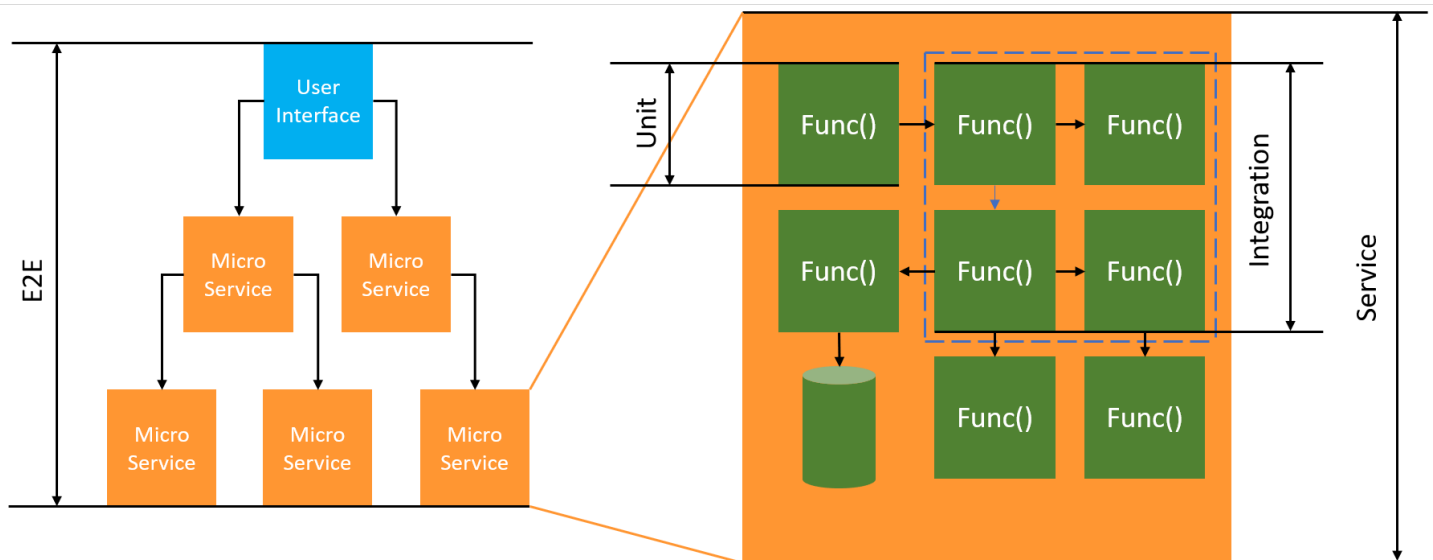
Para probar las funciones sin servidor, se utilizan tipos y técnicas de prueba tradicionales, pero también se debe considerar la posibilidad de probar las aplicaciones sin servidor en conjunto. Las pruebas basadas en la nube proporcionan la medida más precisa de la calidad tanto de las funciones como de las aplicaciones sin servidor.

Una arquitectura de aplicaciones sin servidor incluye servicios administrados que proporcionan las funcionalidades críticas de las aplicaciones mediante llamadas a la API. Por este motivo, el ciclo de desarrollo debe incluir pruebas automatizadas que verifiquen las funcionalidades cuando la función y los servicios interactúen.

Si no crea pruebas basadas en la nube, pueden surgir problemas debido a las diferencias entre su entorno local y el entorno implementado. El proceso de integración continua debe ejecutar pruebas con un conjunto de recursos aprovisionados en la nube antes de promover el código al siguiente entorno de implementación, como el control de calidad, el ensayo o la producción.

Siga leyendo esta breve guía para obtener información sobre las estrategias de prueba para aplicaciones sin servidor, o visite el [repositorio de ejemplos de pruebas sin servidor](#) a fin de profundizar en ejemplos prácticos y específicos para el lenguaje y el tiempo de ejecución de su elección.





Para las pruebas sin servidor, seguirá escribiendo unidades, integraciones y end-to-end pruebas.

- Pruebas unitarias: pruebas que se ejecutan en un bloque de código aislado. Por ejemplo, verificar la lógica empresarial para calcular los gastos de envío en función de un elemento y un destino determinados.
- Pruebas de integración: pruebas en las que participan dos o más componentes o servicios que interactúan, normalmente en un entorno de nube. Por ejemplo, verificar si una función procesa los eventos de una cola.
- End-to-end Pruebas electrónicas: pruebas que verifican el comportamiento en toda la aplicación. Por ejemplo, garantizar que la infraestructura esté configurada correctamente y que los eventos fluyan entre los servicios tal como se espera para registrar el pedido de un cliente.

## Pruebas de aplicaciones sin servidor

Por lo general, utilizará una combinación de métodos para probar el código de sus aplicaciones sin servidor, como las pruebas en la nube, las pruebas con simulaciones y, en ocasiones, las pruebas con emuladores.

### Pruebas en la nube

Las pruebas en la nube son valiosas para todas las fases de las pruebas, incluidas las pruebas unitarias, las pruebas de integración y end-to-end las pruebas. Las pruebas se ejecutan con el código implementado en la nube y se interactúa con los servicios basados en la nube. Este enfoque proporciona la medida más precisa de la calidad del código.

Una forma práctica de depurar la función de Lambda en la nube es a través de la consola con un evento de prueba. Un evento de prueba es una entrada JSON a su función. Si la función no requiere una entrada, el evento puede ser un documento JSON vacío ( {} ). La consola proporciona ejemplos de eventos para una variedad de integraciones de servicios. Tras crear un evento en la consola, puede compartirlo con su equipo para que las pruebas sean más sencillas y coherentes.

### Note

[Probar una función en la consola](#) es una forma rápida de empezar, pero la automatización de los ciclos de prueba garantiza la calidad de la aplicación y la velocidad de desarrollo.

## Herramientas para pruebas

Existen herramientas y técnicas para acelerar los ciclos de retroalimentación del desarrollo. Por ejemplo, tanto [AWS SAM Accelerate](#) como el [modo de vigilancia de AWS CDK](#) reducen el tiempo necesario para actualizar los entornos de nube.

[Moto](#) es una biblioteca de Python que se utiliza para simular servicios y recursos de AWS, de modo que puede probar sus funciones con poca o ninguna modificación mediante el uso de decoradores para interceptar y simular las respuestas.

La característica de validación de [Powertools para AWS Lambda \(Python\)](#) proporciona decoradores para que pueda validar los eventos de entrada y las respuestas de salida de sus funciones de Python.

Para obtener más información, lea la publicación de blog [Prueba unitaria de Lambda con Python y servicios de AWS simulados](#).

Para reducir la latencia relacionada con las iteraciones de implementación en la nube, consulte [AWS Serverless Application Model \(AWS SAM\) Accelerate](#) y el [modo de vigilancia de AWS Cloud Development Kit \(AWS CDK\)](#). Estas herramientas supervisan la infraestructura y el código para detectar cambios. Reaccionan ante estos cambios al crear e implementar automáticamente actualizaciones progresivas en el entorno de nube.

Los ejemplos que utilizan estas herramientas están disponibles en el repositorio de código [Python Test Samples](#).

# Instrumentación del código Python en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas tres bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK para Python](#): un SDK para generar y enviar datos de seguimiento a X-Ray.
- [Powertools para AWS Lambda \(Python\)](#): un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de Powertools para AWS Lambda \(Python\) y AWS SAM para el seguimiento](#)
- [Uso de Powertools para AWS Lambda \(Python\) y el AWS CDK para el seguimiento](#)
- [Uso de ADOT para instrumentar las funciones de Python](#)
- [Uso del X-Ray SDK para instrumentar las funciones de Python](#)
- [Activación del seguimiento con la consola de Lambda](#)
- [Activación del seguimiento con la API de Lambda](#)

- [Activación del seguimiento con AWS CloudFormation](#)
- [Interpretación de un seguimiento de X-Ray](#)
- [Almacenamiento de dependencias de tiempo de ejecución en una capa \(X-Ray SDK\)](#)

## Uso de Powertools para AWS Lambda (Python) y AWS SAM para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Python con módulos integrados de [Powertools para AWS Lambda \(Python\)](#) mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje hello world.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Python 3.9
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

### Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla “Hola, mundo” de Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type Zip --runtime python3.9 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

**Note**

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque el punto de conexión de la API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

7. Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
New XRay Service Graph  
Start time: 2023-02-03 14:59:50+00:00  
End time: 2023-02-03 14:59:50+00:00  
Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -  
Edges: [1]  
Summary_statistics:  
- total requests: 1  
- ok count(2XX): 1  
- error count(4XX): 0  
- fault count(5XX): 0
```

```

- total response time: 0.924
Reference Id: 1 - AWS::Lambda::Function - sam-app>HelloWorldFunction-YBg8yfYt0c9j
- Edges: []
Summary_statistics:
- total requests: 1
- ok count(2XX): 1
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0.016
Reference Id: 2 - client - sam-app>HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app>HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app>HelloWorldFunction-YBg8yfYt0c9j
- 0.739s - Initialization
- 0.016s - Invocation
- 0.013s - ## lambda_handler
- 0.000s - ## app.hello
- 0.000s - Overhead

```

8. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

# Uso de Powertools para AWS Lambda (Python) y el AWS CDK para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Python con módulos integrados de [Powertools para AWS Lambda \(Python\)](#) mediante AWS CDK. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje hello world.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Python 3.9
- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world
cd hello-world
```

2. Inicialice la aplicación.

```
cdk init app --language python
```

3. Instale las dependencias de Python.

```
pip install -r requirements.txt
```

4. Cree un directorio `lambda_function` en la carpeta raíz.

```
mkdir lambda_function
cd lambda_function
```

5. Cree un archivo `app.py` y agregue el siguiente código al archivo. Se trata del código de la función de Lambda.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
                      value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
# ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```



- Abra el directorio `hello_world`. Debería ver un archivo llamado `hello_world_stack.py`.

```
cd ..
cd hello_world
```

- Abra `hello_world_stack.py` y agregue el siguiente código al archivo. Esto contiene el [constructor de Lambda](#), el cual crea la función de Lambda, configura variables de entorno para Powertools y establece la retención de registros en una semana, y el [constructor ApiGatewayV1](#), el cual crea la API de REST.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
            # constructor is in Alpha, o we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
            aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
            docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_9,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
```

```

        handler="app.lambda_handler",
        memory_size=128,
        timeout=Duration.seconds(3),
        architecture=lambda_.Architecture.X86_64,
        environment={
            "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
            "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
            "LOG_LEVEL": "INFO"
        }
    )

    apigw = apigwv1.RestApi(self, "PowertoolsAPI",
        deploy_options=apigwv1.StageOptions(stage_name="dev"))

    hello_api = apigw.root.add_resource("hello")
    hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
        proxy=True))

    CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")

```

## 8. Implementación de la aplicación.

```

cd ..
cdk deploy

```

## 9. Obtenga la URL de la aplicación implementada:

```

aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text

```

## 10. Invoque el punto de conexión de la API:

```

curl -X GET <URL_FROM_PREVIOUS_STEP>

```

Si se realiza de forma correcta, verá el siguiente resultado:

```

{"message":"hello world"}

```

## 11. Para obtener el seguimiento de la función, ejecute [sam traces](#).

```

sam traces

```

El resultado del seguimiento se verá de la siguiente manera:

```

New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -
  Edges: [1]
    Summary_statistics:
      - total requests: 1
      - ok count(2XX): 1
      - error count(4XX): 0
      - fault count(5XX): 0
      - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- Edges: []
  Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
  Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app-HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app-HelloWorldFunction-YBg8yfYt0c9j
  - 0.739s - Initialization
  - 0.016s - Invocation
    - 0.013s - ## lambda_handler
      - 0.000s - ## app.hello
    - 0.000s - Overhead

```

12. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
cdk destroy
```

## Uso de ADOT para instrumentar las funciones de Python

ADOT proporciona [capas](#) de Lambda completamente administradas que empaquetan todo lo necesario para recopilar datos de telemetría mediante el OTel SDK. Utilizando esta capa, se pueden instrumentar las funciones de Lambda sin tener que modificar el código de ninguna función. También se puede configurar la capa para que realice una inicialización personalizada de OTel. Para obtener más información, consulte [Configuración personalizada del recopilador de ADOT en Lambda](#) en la documentación de ADOT.

Para los tiempos de ejecución de Python, puede agregar la capa Lambda administrada por AWS para ADOT Python para instrumentar automáticamente sus funciones. Esta capa funciona para arquitecturas arm64 y x86\_64. Para obtener instrucciones detalladas sobre cómo agregar esta capa, consulte [Soporte de Lambda de AWS Distro for OpenTelemetry para Python](#) en la documentación de ADOT.

## Uso del X-Ray SDK para instrumentar las funciones de Python

Para registrar detalles sobre las llamadas que realiza la función Lambda a otros recursos de la aplicación, también se puede utilizar el AWS X-Ray SDK para Python. Para obtener el SDK, agregue el paquete `aws-xray-sdk` a las dependencias de la aplicación.

Example [requirements.txt](#)

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

En el código de función, puede instrumentar los clientes del SDK de AWS mediante la aplicación de parches a la biblioteca `boto3` con el módulo de `aws_xray_sdk.core`.

Example [función: rastreo de un cliente del SDK de AWS](#)

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

logger = logging.getLogger()
```

```
logger.setLevel(logging.INFO)
patch_all()

client = boto3.client('lambda')
client.get_account_settings()

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
    ...
```

Una vez agregadas las dependencias correctas y realizados los cambios de código necesarios, active el seguimiento en la configuración de la función mediante la consola de Lambda o la API.

## Activación del seguimiento con la consola de Lambda

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

## Activación del seguimiento con la API de Lambda

Configure el rastreo en la función Lambda con AWS CLI o SDK de AWS, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Activación del seguimiento con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

## Interpretación de un seguimiento de X-Ray

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos

sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.



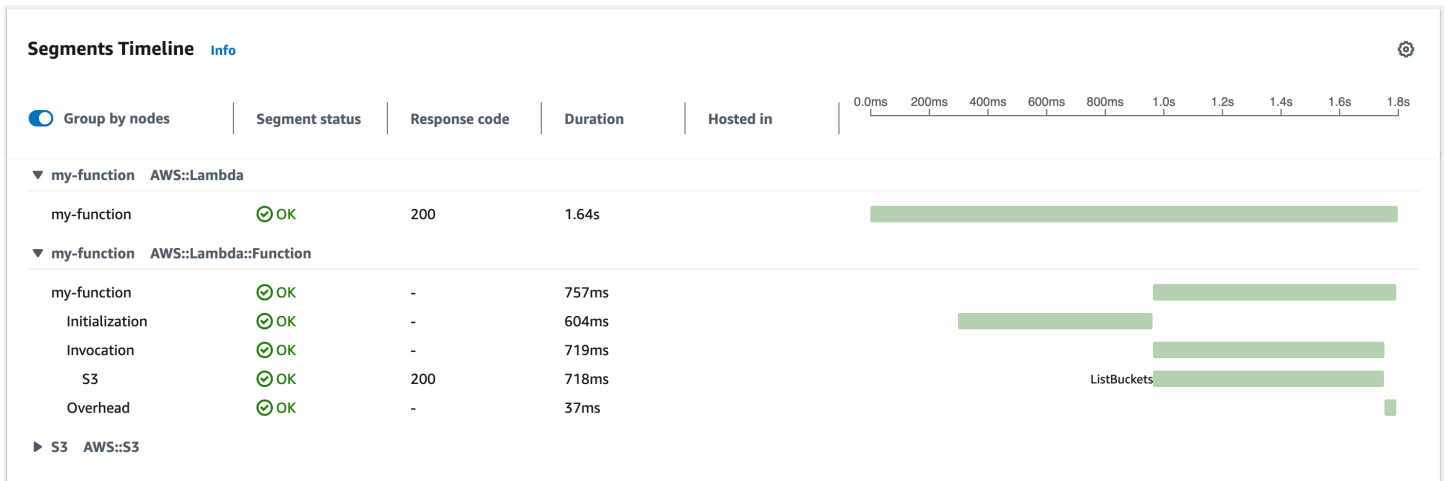
X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo,

se muestra un seguimiento con estos dos segmentos. Ambos se denominan `my-function`, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.



- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte [AWS X-Ray SDK para Python](#) en la Guía para desarrolladores de AWS X-Ray.

### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza cargos por almacenamiento y recuperación del seguimiento. Para más información, consulte [Precios de AWS X-Ray](#).

## Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK)

Si utiliza el X-Ray SDK para instrumentar el código de las funciones de los clientes del SDK de AWS, el paquete de implementación puede llegar a ser bastante grande. Para evitar que se carguen dependencias en tiempo de ejecución cada vez que se actualice el código de las funciones, empaquete el X-Ray SDK en una [capa de Lambda](#).

El siguiente ejemplo muestra un recurso `AWS::Serverless::LayerVersion` que almacena el AWS X-Ray SDK para Python.

Example [template.yml](#): capa de dependencias.

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
```

```
CodeUri: function/.
Tracing: Active
Layers:
  - !Ref libs
  ...
libs:
  Type: AWS::Serverless::LayerVersion
  Properties:
    LayerName: blank-python-lib
    Description: Dependencies for the blank-python sample app.
    ContentUri: package/.
    CompatibleRuntimes:
      - python3.8
```

Con esta configuración, solo se actualiza la capa de la biblioteca si se modifican las dependencias del tiempo de ejecución. Dado que el paquete de implementación de la función contiene únicamente el código, esto puede ayudar a reducir los tiempos de carga.

Para crear una capa de dependencias, es necesario realizar cambios en la compilación para generar el archivo de capas antes de la implementación. Para ver un ejemplo de trabajo, consulte la aplicación de ejemplo [blank-python](#).

# Creación de funciones de Lambda con Ruby

Puede ejecutar código Ruby en AWS Lambda. Lambda ofrece [tiempos de ejecución](#) para Ruby que ejecutan código para procesar eventos. El código se ejecuta en un entorno que incluye AWS SDK for Ruby, con credenciales de un rol de AWS Identity and Access Management (IAM) que usted administre. Para obtener más información sobre las versiones del SDK incluidas en los tiempos de ejecución de Ruby, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Lambda admite los siguientes entornos de tiempos de ejecución de Ruby.

## Ruby

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Ruby 3.3	ruby3.3	Amazon Linux 2023			
Ruby 3.2	ruby3.2	Amazon Linux 2			

Para crear una función Ruby

1. Abra la [consola de Lambda](#).
2. Elija Crear función.
3. Configure los siguientes ajustes:
  - En Nombre de la función: ingrese el nombre de la función.
  - Tiempo de ejecución: elija Ruby 3.2.
4. Elija Crear función.
5. Para configurar un evento de prueba, seleccione Prueba.
6. En Nombre del evento, escriba **test**.
7. Elija Guardar cambios.
8. Elija Test (Probar) para invocar la función.

La consola crea una función de Lambda con un único archivo de origen llamado `lambda_function.rb`. Puede editar este archivo y agregar más archivos en el editor de código integrado. Para guardar los cambios, elija Guardar. A continuación, para ejecutar el código, elija Pruebas.

#### Note

La consola de Lambda utiliza AWS Cloud9 para proporcionar un entorno de desarrollo integrado en el navegador. También puede utilizar AWS Cloud9 para desarrollar funciones de Lambda en su propio entorno. Para obtener más información, consulte [Trabajar con funciones de AWS Lambda utilizando AWS Toolkit](#) en la guía del usuario de AWS Cloud9.

El archivo `lambda_function.rb` exporta una función denominada `lambda_handler` que toma un objeto de evento y un objeto `context`. Esta es la [función de controlador](#) a la que llama Lambda cuando se invoca la función. El tiempo de ejecución de la función de Ruby obtiene eventos de Lambda y los pasa al controlador. En la configuración de función, el valor de controlador es `lambda_function.lambda_handler`.

Al guardar el código de función, la consola de Lambda crea un paquete de implementación de archivo `.zip`. Cuando desarrolle el código de función fuera de la consola (mediante un IDE), debe [crear un paquete de implementación](#) para cargar el código a la función de Lambda.

#### Note

Para comenzar con el desarrollo de aplicaciones en su entorno local, implemente una de las aplicaciones de ejemplo disponibles en el repositorio de GitHub de esta guía.

Aplicaciones de Lambda de ejemplo en Ruby

- [blank-ruby](#): una función de Ruby que muestra el uso del registro, las variables de entorno, el seguimiento de AWS X-Ray, las pruebas de unidad y el AWS SDK.
- [Ejemplos de código de Ruby para AWS Lambda](#): muestras de código escritas en Ruby que demuestran cómo interactuar con AWS Lambda.

El tiempo de ejecución de la función pasa un objeto `context` al controlador, además del evento de invocación. El [objeto context](#) contiene información adicional acerca de la invocación, la función y el entorno de ejecución. Hay más información disponible en las variables de entorno.

Su función de Lambda tiene un grupo de registros de Registros de CloudWatch. El tiempo de ejecución de la función envía detalles de cada invocación a Registros de CloudWatch. Se transmite cualquier [registro que su función genere](#) durante la invocación. Si su función devuelve un error, Lambda formatea el error y lo devuelve al invocador.

## Temas

- [Versiones del SDK incluidas en el tiempo de ejecución](#)
- [Habilitación de otro JIT de Ruby \(YJIT\)](#)
- [Definir el controlador de la función de Lambda en Ruby](#)
- [Implementar funciones de Lambda de Ruby con archivos .zip](#)
- [Implementación de funciones de Lambda de Ruby con imágenes de contenedor](#)
- [Uso del objeto de contexto Lambda para recuperar la información de la función de Ruby](#)
- [Registro y supervisión de las funciones de Lambda de Ruby](#)
- [Instrumentación del código Ruby en AWS Lambda](#)

## Versiones del SDK incluidas en el tiempo de ejecución

La versión del AWS SDK incluida en el tiempo de ejecución de Ruby depende de la versión del tiempo de ejecución y de su Región de AWS. AWS SDK para Ruby está diseñado para ser modular y está separado por Servicio de AWS. Para encontrar el número de versión de una gema de servicio concreta incluida en el tiempo de ejecución que está utilizando, cree una función de Lambda con código en el siguiente formato. Sustituya `aws-sdk-s3` y `Aws::S3` por el nombre de las gemas de servicio que utilice su código.

```
require 'aws-sdk-s3'

def lambda_handler(event:, context:)
  puts "Service gem version: #{Aws::S3::GEM_VERSION}"
  puts "Core version: #{Aws::CORE_GEM_VERSION}"
end
```

## Habilitación de otro JIT de Ruby (YJIT)

El tiempo de ejecución de Ruby 3.2 es compatible con [YJIT](#), un compilador JIT de Ruby ligero y minimalista. YJIT proporciona un rendimiento significativamente mayor, pero también utiliza más memoria que el intérprete de Ruby. Se recomienda YJIT para cargas de trabajo de Ruby on Rails.

YJIT no está habilitado de forma predeterminada. Para habilitar YJIT para una función de Ruby 3.2, establezca la variable de entorno `RUBY_YJIT_ENABLE` en 1. Para confirmar que YJIT está habilitado, imprima el resultado del método `RubyVM::YJIT.enabled?`.

Example — Confirme que YJIT está habilitado

```
puts(RubyVM::YJIT.enabled?())  
# => true
```

# Definir el controlador de la función de Lambda en Ruby

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

## Temas

- [Concepts básicos del controlador de Ruby](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Ruby](#)

## Concepts básicos del controlador de Ruby

En el siguiente ejemplo, el archivo `function.rb` define un método de controlador llamado `handler`. La función de controlador tiene dos objetos como entrada y devuelve un documento JSON.

### Example function.rb

```
require 'json'

def handler(event:, context:)
  { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

En su configuración de la función, el valor `handler` indica a Lambda dónde encontrar el controlador. Para el ejemplo anterior, el valor correcto para este valor es **`function.handler`**. Incluye dos nombres separados por un punto: el nombre del archivo y el nombre del método del controlador.

También puede definir su método de controlador en una clase. En el siguiente ejemplo se define un método de controlador llamado `process` en una clase llamada `Handler` en un módulo llamado `LambdaFunctions`.

### Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:, context:)
      "Hello!"
    end
  end
end
```

```
end
end
end
```

En este caso, el valor del controlador es **`source.LambdaFunctions::Handler.process`**.

Los dos objetos que el controlador acepta son el evento de invocación y el contexto. El evento es un objeto Ruby que contiene la carga que proporciona el invocador. Si la carga es un documento JSON, el objeto de evento es un hash de Ruby. De lo contrario, es una cadena. Este [objeto de contexto](#) tiene métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución.

El controlador de la función se ejecuta cada vez que se invoca la función de Lambda. El código estático fuera del controlador se ejecuta una vez por instancia de la función. Si su controlador utiliza recursos como clientes de SDK y conexiones de bases de datos, puede crearlos fuera del método de controlador para volver a utilizarlos en varias invocaciones.

Cada instancia de la función permite procesar varios eventos de invocación, pero solo procesa un evento cada vez. El número de instancias que procesan un evento en un momento dado es la simultaneidad de la función. Para obtener más información acerca del entorno de ejecución de Lambda, consulte [Comprender el ciclo de vida del entorno de ejecución de Lambda](#).

## Prácticas recomendadas de codificación para las funciones de Lambda en Ruby

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad. Por ejemplo, en Ruby, podría tener este aspecto:

```
def lambda_handler(event:, context:)
  foo = event['foo']
  bar = event['bar']

  result = my_lambda_function(foo:, bar:)
end

def my_lambda_function(foo:, bar:)
```



```
// MyLambdaFunction logic here  
  
end
```

- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. En el caso del entorno de ejecución de Ruby, estas incluyen el SDK de AWS. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación. En las funciones creadas en Ruby, evite cargar toda la biblioteca de SDK de AWS como parte del paquete de implementación. En lugar de ello, cree dependencias selectivas de las gemas que seleccionen los componentes del SDK que necesita (por ejemplo, las gemas de SDK de Amazon S3 o DynamoDB).
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).

- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Implementar funciones de Lambda de Ruby con archivos .zip

El código de la función de AWS Lambda incluye un archivo .rb que contiene el código del controlador de la función, junto con las dependencias adicionales (gemas) de las que dependa el código. Para implementar este código de función en Lambda, se utiliza un paquete de despliegue. Este paquete puede ser un archivo de archivos .zip o una imagen de contenedor. Para obtener más información sobre el uso de imágenes de contenedores con Ruby, consulte [Implementar funciones de Lambda Ruby con imágenes de contenedores](#).

Para crear un paquete de despliegue como un archivo de archivos .zip, puede emplear utilidad de archivado de archivos .zip incorporada de la herramienta de línea de comandos, o cualquier otra utilidad de archivos .zip, como [7zip](#). En los ejemplos que se muestran en las siguientes secciones se supone que está utilizando una herramienta zip de línea de comandos en un entorno Linux o macOS. Para utilizar los mismos comandos en Windows, puede [instalar el Subsistema de Windows para Linux](#) a fin de obtener una versión de Ubuntu y Bash integrada con Windows.

Tenga en cuenta que Lambda utiliza permisos de archivo de POSIX, por lo que puede necesitar [establecer permisos para la carpeta del paquete de despliegue](#) antes de crear el archivo de archivos .zip.

En los comandos de ejemplo de las siguientes secciones, se utiliza la utilidad [Agrupador](#) para agregar dependencias al paquete de despliegue. Ejecute el siguiente comando para instalar el agrupador.

```
gem install bundler
```

## Secciones

- [Dependencias en Ruby](#)
- [Creación de un paquete de despliegue .zip sin dependencias](#)
- [Creación de un paquete de despliegue .zip con dependencias](#)
- [Creación de una capa de Ruby para las dependencias](#)
- [Creación de paquetes de despliegue .zip con bibliotecas nativas](#)
- [Creación y actualización de funciones de Lambda Ruby mediante archivos .zip](#)

## Dependencias en Ruby

Para las funciones de Lambda que utilizan el tiempo de ejecución de Ruby, una dependencia puede ser cualquier gema de Ruby. Cuando implemente la función mediante un archivo .zip, podrá agregar estas dependencias a su archivo .zip con el código de la función o utilizar una capa de Lambda. Una capa es un archivo .zip independiente que puede contener código adicional y otro contenido. Para obtener más información sobre el uso de las capas de Lambda, consulte [Capas de Lambda](#).

En el tiempo de ejecución de Ruby se incluye el AWS SDK for Ruby. Si la función utiliza el SDK, no necesita agruparlo con el código. Sin embargo, para mantener el control total de las dependencias o para utilizar una versión específica del SDK, puede agregarla al paquete de despliegue de la función. Puede incluir el SDK en el archivo .zip o agregarlo mediante una capa de Lambda. Las dependencias del archivo .zip o de las capas de Lambda tienen prioridad sobre las versiones incluidas en el tiempo de ejecución. Para saber qué versión del SDK para Ruby está incluida en la versión del tiempo de ejecución, consulte [the section called “Versiones del SDK incluidas en el tiempo de ejecución”](#).

Según el [modelo de responsabilidad compartida de AWS](#), usted es responsable de la administración de cualquier dependencia en los paquetes de despliegue de sus funciones. Esto incluye la aplicación de actualizaciones y parches de seguridad. Para actualizar las dependencias del paquete de despliegue de la función, primero cree un nuevo archivo .zip y, a continuación, cárguelo en Lambda. Para obtener más información, consulte [Creación de un paquete de despliegue .zip con dependencias](#) y [Creación y actualización de funciones de Lambda Ruby mediante archivos .zip](#).

### Creación de un paquete de despliegue .zip sin dependencias

Si el código de la función no tiene dependencias, el archivo .zip solo contiene el archivo .rb con el código del controlador de la función. Utilice la utilidad de compresión que prefiera para crear un archivo .zip con el archivo .rb en la raíz. Si el archivo .rb no está en la raíz del archivo .zip, Lambda no podrá ejecutar el código.

Para obtener información sobre cómo implementar un archivo .zip para crear una nueva función de Lambda o actualizar una existente, consulte [Creación y actualización de funciones de Lambda Ruby mediante archivos .zip](#).

### Creación de un paquete de despliegue .zip con dependencias

Si el código de la función depende de gemas de Ruby adicionales, puede agregar estas dependencias al archivo .zip con el código de la función o utilizar una [capa de Lambda](#). En

las instrucciones de esta sección, se muestra cómo incluir las dependencias en el paquete de despliegue .zip. Para obtener instrucciones sobre cómo incluir las dependencias en una capa, consulte [the section called “Creación de una capa de Ruby para las dependencias”](#).

Supongamos que el código de la función se guarda en un archivo llamado `lambda_function.rb` del directorio del proyecto. Los siguientes comandos de la CLI crean un archivo .zip llamado `my_deployment_package.zip` que contiene el código de la función de y sus dependencias.

### Creación del paquete de implementación

1. En el directorio del proyecto, cree un Gemfile para especificar las dependencias.

```
bundle init
```

2. Con el editor de texto que prefiera, edite el Gemfile para especificar las dependencias de la función. Por ejemplo, para utilizar la gema TZInfo, edite el Gemfile para que tenga una apariencia similar a la siguiente.

```
source "https://rubygems.org"  
gem "tzinfo"
```

3. Ejecute el siguiente comando para instalar las gemas especificadas en el Gemfile del directorio del proyecto. Este comando configura `vendor/bundle` como la ruta predeterminada para las instalaciones de gemas.

```
bundle config set --local path 'vendor/bundle' && bundle install
```

Debería ver un resultado similar a este.

```
Fetching gem metadata from https://rubygems.org/.....  
Resolving dependencies...  
Using bundler 2.4.13  
Fetching tzinfo 2.0.6  
Installing tzinfo 2.0.6  
...
```

#### Note

Para volver a instalar gemas a nivel global más adelante, ejecute el siguiente comando.

```
bundle config set --local system 'true'
```

4. Cree un archivo `.zip` que contenga el archivo `lambda_function.rb` con el código del controlador de la función y las dependencias que instaló en el paso anterior.

```
zip -r my_deployment_package.zip lambda_function.rb vendor
```

Debería ver un resultado similar a este.

```
adding: lambda_function.rb (deflated 37%)
  adding: vendor/ (stored 0%)
  adding: vendor/bundle/ (stored 0%)
  adding: vendor/bundle/ruby/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/build_info/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

## Creación de una capa de Ruby para las dependencias

En las instrucciones de esta sección, se muestra cómo incluir las dependencias en una capa. Para obtener instrucciones sobre cómo incluir las dependencias en el paquete de implementación, consulte [the section called “Creación de un paquete de despliegue .zip con dependencias”](#).

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio `/opt` de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable `PATH` ya incluye rutas de carpeta específicas en el directorio `/opt`. Para garantizar que la variable `PATH` recoja el contenido de la capa, el archivo `.zip` de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

- `ruby/gems/2.7.0` (`GEM_PATH`)
- `ruby/lib` (`RUBYLIB`)

Por ejemplo, la estructura del archivo `.zip` de la capa podría tener el siguiente aspecto:

```
json.zip
```

```
# ruby/gems/2.7.0/  
  | build_info  
  | cache  
  | doc  
  | extensions  
  | gems  
  | # json-2.1.0  
# specifications  
  # json-2.1.0.gemspec
```

Además, Lambda detecta de forma automática cualquier biblioteca en el directorio `/opt/lib` y todos los archivos binarios en el directorio `/opt/bin`. Para asegurarse de que Lambda encuentre el contenido de la capa de forma correcta, también puede crear una capa con la siguiente estructura:

```
custom-layer.zip  
# lib  
  | lib_1  
  | lib_2  
# bin  
  | bin_1  
  | bin_2
```

Después de empaquetar la capa, consulte [the section called “Creación y eliminación de capas”](#) y [the section called “Adición de capas”](#) para completar la configuración de la capa.

## Creación de paquetes de despliegue .zip con bibliotecas nativas

Muchas gemas de Ruby comunes, como `nokogiri`, `nio4r` y `mysql`, contienen extensiones nativas escritas en C. Cuando agregue bibliotecas que contienen código C al paquete de despliegue, deberá compilar el paquete correctamente para garantizar que sea compatible con el entorno de ejecución de Lambda.

Para las aplicaciones de producción, recomendamos crear e implementar el código mediante AWS Serverless Application Model (AWS SAM). En AWS SAM, utilice la opción `sam build --use-container` para crear la función dentro de un contenedor de Docker tipo Lambda. Para obtener más información sobre el uso de AWS SAM para implementar el código de la función, consulte [Creación de aplicaciones](#) en la Guía para desarrolladores de AWS SAM.

Para crear un paquete de despliegue .zip que contenga gemas con extensiones nativas sin utilizar AWS SAM, también puede utilizar un contenedor para agrupar las dependencias en un entorno que sea igual al entorno de tiempo de ejecución de Ruby Lambda. Para completar estos pasos,

debe tener Docker instalado en su equipo de compilación. Para obtener más información sobre la instalación de Docker, consulte [Instalar Docker Engine](#).

Para crear un paquete de despliegue .zip en un contenedor de Docker

1. Cree una carpeta en el equipo de compilación local para guardar el contenedor. Dentro de esa carpeta, cree un archivo llamado `dockerfile` y pegue el siguiente código en él.

```
FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
RUN gem update bundler
CMD "/bin/bash"
```

2. Dentro de la carpeta en la que creó el `dockerfile`, ejecute el siguiente comando para crear el contenedor de Docker.

```
docker build -t awsruby32 .
```

3. Desplácese hasta el directorio del proyecto que contiene el archivo `.rb` con el código del controlador de la función y el `Gemfile` para especificar las dependencias de la función. Desde dentro de ese directorio, ejecute el siguiente comando para iniciar el contenedor de Ruby Lambda.

Linux/MacOS

```
docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
```

#### Note

En macOS, es posible que vea una advertencia que informa que la plataforma de la imagen solicitada no coincide con la plataforma host detectada. Ignore esta advertencia.

Windows PowerShell

```
docker run --rm -it -v ${pwd}:/var/task -w /var/task awsruby32
```

Cuando se inicie el contenedor, debería ver un mensaje de bash.



```
bash-4.2#
```

- Configure la agrupación de paquetes para instalar las gemas especificadas en el Gemfile del directorio `vendor/bundle` local e instale las dependencias.

```
bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
```

- Cree un paquete de despliegue con el código de la función y sus dependencias. En este ejemplo, el archivo que contiene el código del controlador de la función se llama `lambda_function.rb`.

```
bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
```

- Salga del contenedor y regrese al directorio del proyecto local.

```
bash-4.2# exit
```

Ahora puede utilizar el paquete de despliegue del archivo `.zip` para crear o actualizar la función de Lambda. Consulte [Creación y actualización de funciones de Lambda Ruby mediante archivos .zip](#)

## Creación y actualización de funciones de Lambda Ruby mediante archivos .zip

Una vez que haya creado su paquete de despliegue `.zip`, puede utilizarlo para crear una nueva función de Lambda o actualizar una existente. Puede implementar el paquete `.zip` a través de la consola, la AWS Command Line Interface y la API de Lambda. También puede crear y actualizar funciones de Lambda mediante AWS Serverless Application Model (AWS SAM) y AWS CloudFormation.

El tamaño máximo de un paquete de despliegue `.zip` para Lambda es de 250 MB (descomprimido). Tenga en cuenta que este límite se aplica al tamaño combinado de todos los archivos que cargue, incluidas las capas de Lambda.

El tiempo de ejecución de Lambda necesita permiso para leer los archivos del paquete de implementación. En la notación octal de permisos de Linux, Lambda necesita 644 permisos para archivos no ejecutables (`rw-r--r--`) y 755 permisos (`rw-r-xr-x`) para directorios y archivos ejecutables.

En Linux y macOS, utilice el comando `chmod` para cambiar los permisos de los archivos y directorios del paquete de implementación. Por ejemplo, para brindarle a un archivo ejecutable los permisos correctos, ejecute el siguiente comando.

```
chmod 755 <filepath>
```

Para cambiar los permisos de los archivos en Windows, consulte [Set, View, Change, or Remove Permissions on an Object](#) en la documentación de Microsoft Windows.

## Creación y actualización de funciones con archivos .zip mediante la consola

Para crear una nueva función, primero debe crearla en la consola y, a continuación, cargar el archivo .zip. Para actualizar una función existente, abra la página de la función correspondiente y, a continuación, siga el mismo procedimiento para agregar el archivo .zip actualizado.

Si el archivo .zip tiene un tamaño inferior a 50 MB, puede crear o actualizar una función al cargarlo directamente desde su equipo local. Para archivos .zip de más de 50 MB, primero debe cargar su paquete en un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS Management Console, consulte [Introducción a Amazon S3](#). Para cargar archivos mediante la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

### Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

Para crear una nueva función (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija Crear función.
2. Elija Crear desde cero.
3. En Información básica, haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función.
  - b. En Tiempo de ejecución, seleccione el tiempo de ejecución que desea utilizar.

- c. (Opcional) Para Arquitectura, elija la arquitectura del conjunto de instrucciones para su función. La arquitectura predeterminada es x86\_64. Asegúrese de que el paquete de despliegue .zip para su función sea compatible con la arquitectura del conjunto de instrucciones que seleccione.
4. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o utilizar uno existente.
5. Elija Crear función. Lambda crea una función básica “Hola, mundo” mediante el tiempo de ejecución elegido.

Para cargar un archivo .zip desde su equipo local (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar el archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija un archivo .zip.
5. Para cargar el archivo .zip, haga lo siguiente:
  - a. Seleccione Cargar y, a continuación, seleccione su archivo .zip en el selector de archivos.
  - b. Elija Abrir.
  - c. Seleccione Guardar.

Carga de un archivo .zip desde un bucket de Amazon S3 (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar un nuevo archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija la ubicación de Amazon S3.
5. Pegue la URL del enlace de Amazon S3 de su archivo .zip y seleccione Guardar.

## Actualización de las funciones del archivo .zip mediante el editor de código de la consola

Para algunas funciones con paquetes de despliegue en formato .zip, puede utilizar el editor de código integrado en la consola de Lambda para actualizar el código de la función de forma directa. Para utilizar esta característica, la función debe cumplir los siguientes criterios:

- La función debe utilizar uno de los tiempos de ejecución del lenguaje interpretado (Python, Node.js o Ruby)
- El paquete de despliegue de la función debe tener un tamaño inferior a 3 MB.

El código de función de las funciones con paquetes de despliegue de imágenes de contenedores no se puede editar directamente en la consola.

Para actualizar el código de función mediante el editor de código de la consola

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, seleccione su archivo de código fuente y edítelo en el editor de código integrado.
4. Cuando haya terminado de editar el código, seleccione Implementar para guardar los cambios y actualizar la función.

## Creación y actualización de funciones con archivos .zip mediante la AWS CLI

Puede utilizar la [AWS CLI](#) para crear una nueva función o actualizar una existente con un archivo .zip. Utilice los comandos [create-function](#) y [update-function-code](#) para implementar su paquete .zip. Si el archivo .zip tiene un tamaño inferior a 50 MB, puede cargarlo desde una ubicación de archivo en su equipo de compilación local. Para archivos más grandes, debe cargar su paquete .zip desde un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

**Note**

Si carga su archivo .zip desde un bucket de Amazon S3 con la AWS CLI, el bucket debe estar ubicado en la misma Región de AWS que su función.

Para crear una nueva función mediante un archivo .zip con la AWS CLI, debe especificar lo siguiente:

- El nombre de la función (`--function-name`).
- El tiempo de ejecución de la función (`--runtime`).
- El nombre de recurso de Amazon (ARN) del [rol de ejecución](#) de la función (`--role`).
- El nombre del método de controlador en el código de la función (`--handler`).

También debe especificar la ubicación del archivo .zip. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda create-function --function-name myFunction \  
--runtime ruby3.2 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice la opción `--code`, como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `S3ObjectVersion` para los objetos con versiones.

```
aws lambda create-function --function-name myFunction \  
--runtime ruby3.2 --handler lambda_function.lambda_handler \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Para actualizar una función existente mediante la CLI, especifique el nombre de la función mediante el parámetro `--function-name`. También debe especificar la ubicación del archivo .zip que desea utilizar para actualizar el código de la función. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice las opciones `--s3-bucket` y `--s3-key` tal como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `--s3-object-version` para los objetos con versiones.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Creación y actualización de funciones con archivos .zip mediante la API de Lambda

Para crear y actualizar funciones con un archivo de archivos .zip, utilice las siguientes operaciones de la API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creación y actualización de funciones con archivos .zip mediante AWS SAM

AWS Serverless Application Model (AWS SAM) es un conjunto de herramientas que ayuda a agilizar el proceso de creación y ejecución de aplicaciones sin servidor en AWS. Defina los recursos de su aplicación en una plantilla YAML o JSON y utilice la interfaz de la línea de comandos de AWS SAM (AWS SAM CLI) para crear, empaquetar e implementar sus aplicaciones. Al crear una función de Lambda a partir de una plantilla de AWS SAM, AWS SAM crea automáticamente un paquete de despliegue .zip o una imagen de contenedor con el código de la función y las dependencias que especifique. Para obtener más información sobre el uso de AWS SAM para crear e implementar funciones de Lambda, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

También puede utilizar AWS SAM para crear una función de Lambda con un archivo de archivos .zip existente. Para crear una función de Lambda mediante AWS SAM, puede guardar el archivo .zip en un bucket de Amazon S3 o en una carpeta local de su equipo de compilación. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS SAM, el recurso `AWS::Serverless::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos `.zip`:

- `PackageType`: se establece como `Zip`.
- `CodeUri`: se establece como el URI de Amazon S3, la ruta a la carpeta local o el objeto [FunctionCode](#) del código de la función.
- `Runtime`: se establece como el entorno de ejecución elegido

Con AWS SAM, si su archivo `.zip` tiene más de 50 MB, no es necesario cargarlo primero en un bucket de Amazon S3. AWS SAM puede cargar paquetes `.zip` hasta el tamaño máximo permitido de 250 MB (descomprimidos) desde una ubicación de su equipo de compilación local.

Para obtener más información sobre la implementación de funciones mediante un archivo `.zip` en AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS SAM.

## Creación y actualización de funciones con archivos `.zip` mediante AWS CloudFormation

Puede utilizar AWS CloudFormation para crear una función de Lambda con un archivo de archivos `.zip`. Para crear una función de Lambda a partir de un archivo `.zip`, primero debe cargar el archivo a un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS CloudFormation, el recurso `AWS::Lambda::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos `.zip`:

- `PackageType`: se establece como `Zip`.
- `Code`: ingrese el nombre del bucket de Amazon S3 y el nombre del archivo `.zip` en los campos `S3Bucket` y `S3Key`.
- `Runtime`: se establece como el tiempo de ejecución elegido.

El archivo `.zip` que genera AWS CloudFormation no puede superar los 4 MB. Para obtener más información sobre la implementación de funciones mediante un archivo `.zip` en AWS CloudFormation, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

# Implementación de funciones de Lambda de Ruby con imágenes de contenedor

Hay tres formas de crear una imagen de contenedor para una función de Lambda en Ruby:

- [Uso de una imagen base de AWS para Ruby](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Ruby](#) en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Ruby](#) en la imagen.

## Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.



## Temas

- [Imágenes base de AWS para Ruby](#)
- [Uso de una imagen base de AWS para Ruby](#)
- [Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución](#)

## Imágenes base de AWS para Ruby

AWS proporciona las siguientes imágenes base para Ruby:

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
3.3	Ruby 3.3	Amazon Linux 2023	<a href="#">Dockerfile para Ruby 3.3 en GitHub</a>	
3.2	Ruby 3.2	Amazon Linux 2	<a href="#">Dockerfile para Ruby 3.2 en GitHub</a>	

Repositorio de Amazon ECR: [gallery.ecr.aws/lambda/ruby](https://gallery.ecr.aws/lambda/ruby)

## Uso de una imagen base de AWS para Ruby

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#)
- Ruby

### Creación de una imagen a partir de una imagen base

#### Cómo crear una imagen de contenedor para Ruby

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
```

```
cd example
```

2. Cree un nuevo archivo denominado `Gemfile`. Aquí se enumeran los paquetes de RubyGems necesarios para la aplicación. AWS SDK for Ruby está disponible en RubyGems. Debe elegir archivos gem de servicios de AWS específicos para instalar. Por ejemplo, para utilizar la [gema de Ruby para Lambda](#), su Gemfile debe tener este aspecto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

Como alternativa, el archivo gem [aws-sdk](#) contiene todos los archivos gem de los servicios de AWS disponibles. Este archivo gem es muy grande. Le recomendamos que lo utilice solo si depende de muchos servicios de AWS.

3. Instale las dependencias especificadas en el Gemfile mediante el [paquete de instalación](#).

```
bundle install
```

4. Cree un nuevo archivo denominado `lambda_function.rb`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

### Example Función de Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:, context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Cree un nuevo Dockerfile. El siguiente Dockerfile de ejemplo utiliza una [imagen base de AWS](#). Este Dockerfile utiliza la siguiente configuración:
  - Establezca la propiedad FROM en el URI de la imagen base.
  - Utilice el comando COPY para copiar el código de la función y las dependencias del tiempo de ejecución a `{LAMBDA_TASK_ROOT}`, una [variable de entorno definido de Lambda](#).
  - Establezca el argumento CMD para el controlador de la función de Lambda.

## Example Dockerfile

```
FROM public.ecr.aws/lambda/ruby:3.2

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler (could also be done as a parameter override outside
of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

### (Opcional) Prueba de la imagen de forma local

1. Inicie la imagen de Docker con el comando `docker run`. En este ejemplo, `docker-image` es el nombre de la imagen y `test` es la etiqueta.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

2. Desde una nueva ventana de terminal, publique un evento en el punto de conexión local.

#### Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

#### PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
"application/json"
```

3. Obtenga el ID del contenedor.

```
docker ps
```

4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en

Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

## Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución

Si usa una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir el cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución extiende el [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de la función.

Instale el [cliente de interfaz de tiempo de ejecución de Lambda para Ruby](#) mediante el administrador de paquetes de RubyGems.org:

```
gem install aws_lambda_ri
```

También puede descargar el [cliente de interfaz de tiempo de ejecución de Ruby](#) desde GitHub. El cliente de interfaz de tiempo de ejecución admite las versiones 2.5.x a 2.7.x de Ruby.

En el siguiente ejemplo, se muestra cómo crear una imagen de contenedor para Ruby con una imagen base que no es de AWS. El Dockerfile de ejemplo utiliza una imagen base oficial de Ruby. El Dockerfile incluye el cliente de interfaz de tiempo de ejecución.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [Docker](#)
- Ruby

### Creación de imágenes a partir de una imagen base alternativa

Para crear una imagen de contenedor para Ruby a partir de una imagen base alternativa

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example
```



```
cd example
```

2. Cree un nuevo archivo denominado `Gemfile`. Aquí se enumeran los paquetes de RubyGems necesarios para la aplicación. AWS SDK for Ruby está disponible en RubyGems. Debe elegir archivos gem de servicios de AWS específicos para instalar. Por ejemplo, para utilizar la [gema de Ruby para Lambda](#), su Gemfile debe tener este aspecto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

Como alternativa, el archivo gem [aws-sdk](#) contiene todos los archivos gem de los servicios de AWS disponibles. Este archivo gem es muy grande. Le recomendamos que lo utilice solo si depende de muchos servicios de AWS.

3. Instale las dependencias especificadas en el Gemfile mediante el [paquete de instalación](#).

```
bundle install
```

4. Cree un nuevo archivo denominado `lambda_function.rb`. Puede agregar el siguiente código de función de muestra al archivo para realizar pruebas o utilizar su propio código.

### Example Función de Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:, context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Cree un nuevo Dockerfile. El siguiente Dockerfile utiliza una imagen base de Ruby en lugar de una [imagen base de AWS](#). El Dockerfile incluye el [cliente de interfaz de tiempo de ejecución para Ruby](#), que hace que la imagen sea compatible con Lambda. Como alternativa, puede agregar el cliente de interfaz de tiempo de ejecución al Gemfile de su aplicación.
  - Establezca la propiedad FROM como la imagen base de Ruby.
  - Cree un directorio para el código de la función y una variable de entorno que apunte a ese directorio. En este ejemplo, el directorio es `/var/task`, que replica el entorno de ejecución

de Lambda. Sin embargo, puede elegir cualquier directorio para el código de la función porque el Dockerfile no utiliza una imagen base AWS.

- Configure ENTRYPOINT como el módulo que desea que el contenedor de Docker ejecute cuando se inicie. En este caso, el módulo es el cliente de interfaz de tiempo de ejecución.
- Establezca el argumento CMD para el controlador de la función de Lambda.

### Example Dockerfile

```
FROM ruby:2.7

# Install the runtime interface client for Ruby
RUN gem install aws_lambda_ri

# Add the runtime interface client to the PATH
ENV PATH="/usr/local/bundle/bin:${PATH}"

# Create a directory for the Lambda function
ENV LAMBDA_TASK_ROOT=/var/task
RUN mkdir -p ${LAMBDA_TASK_ROOT}
WORKDIR ${LAMBDA_TASK_ROOT}

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "aws_lambda_ri" ]

# Set the CMD to your handler (could also be done as a parameter override outside
of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

(Opcional) Prueba de la imagen de forma local

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. Puede [crear el emulador en su imagen](#) o usar el procedimiento siguiente para instalarlo en su equipo local.

Para instalar y ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Desde el directorio del proyecto, ejecute el siguiente comando para descargar el emulador de interfaz de tiempo de ejecución (arquitectura x86-64) de GitHub e instalarlo en su equipo local.

Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Para instalar el emulador arm64, reemplace la URL del repositorio de GitHub en el comando anterior por lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"  
if (-not (Test-Path $dirPath)) {
```

```

    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath

```

Para instalar el emulador arm64, reemplace el `$downloadLink` con lo siguiente:

```

https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64

```

2. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:

- `docker-image` es el nombre de la imagen y `test` es la etiqueta.
- `aws_lambda_rie lambda_function.LambdaFunction::Handler.process` es el ENTRYPOINT seguido del CMD de su Dockerfile.

### Linux/macOS

```

docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  aws_lambda_rie lambda_function.LambdaFunction::Handler.process

```

### PowerShell

```

docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
  --entrypoint /aws-lambda/aws-lambda-rie `
  docker-image:test `
  aws_lambda_rie lambda_function.LambdaFunction::Handler.process

```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

**Note**

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

3. Publique un evento en el punto de conexión local.

**Linux/macOS**

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{"payload": "hello world!"}'
```

**PowerShell**

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload": "hello world!"}' -ContentType "application/json"
```

4. Obtenga el ID del contenedor.

```
docker ps
```

5. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{  
  "repository": {
```

```
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.

7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.



# Uso del objeto de contexto Lambda para recuperar la información de la función de Ruby

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución.

## Métodos de context

- `get_remaining_time_in_millis`: devuelve el número de milisegundos que quedan antes del tiempo de espera de la ejecución.

## Propiedades de context

- `function_name`: el nombre de la función de Lambda.
- `function_version`: la [versión](#) de la función.
- `invoked_function_arn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `memory_limit_in_mb`: cantidad de memoria asignada a la función.
- `aws_request_id`: el identificador de la solicitud de invocación.
- `log_group_name`: grupo de registros de para la función.
- `log_stream_name`: el flujo de registro de la instancia de la función.
- `deadline_ms`: la fecha en la que la función agota su tiempo de ejecución en milisegundos de tiempo Unix.
- `identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
- `client_context`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.

# Registro y supervisión de las funciones de Lambda de Ruby

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre y envía registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación al flujo de registro y retransmite los registros y otras salidas desde el código de la función. Para obtener más información, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Esta página describe cómo producir resultados de registro a partir del código de la función de Lambda o registros de acceso mediante AWS Command Line Interface, la consola de Lambda o la consola de CloudWatch.

## Secciones

- [Crear una función que devuelve registros](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)
- [Trabajo con la biblioteca de registradores en Ruby](#)

## Crear una función que devuelve registros

Para generar registros desde el código de su función puede utilizar instrucciones `puts` o cualquier biblioteca de registro que escriba en `stdout` o en `stderr`. En el siguiente ejemplo, se registran los valores de las variables de entorno y el objeto de evento.

Example `lambda_function.rb`

```
# lambda_function.rb

def handler(event:, context:)
  puts "## ENVIRONMENT VARIABLES"
  puts ENV.to_a
  puts "## EVENT"
  puts event.to_a
end
```

## Example formato de registro

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
  'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c',
  'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Duration: 15.74 ms Billed
  Duration: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmpl1f1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85
  Sampled: true
```

El tiempo de ejecución de Ruby registra las líneas START, END y REPORT de cada invocación. La línea del informe proporciona los siguientes detalles.

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.
- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- TraceId de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

Para obtener registros más detallados, utilice [the section called “Trabajo con la biblioteca de registradores en Ruby”](#).

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

### Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBu1QgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

### Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
```

```
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad base64 está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar base64 -D.

### Example get-logs.sh script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza sed para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando get-log-events.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como get-logs.sh.

La opción cli-binary-format es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute aws configure set cli-binary-format raw-in-base64-out. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

### Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

### Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
}
```

```
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

## Trabajo con la biblioteca de registradores en Ruby

La [biblioteca de logger](#) de Ruby devuelve registros optimizados que se leen fácilmente. Utilice la utilidad de logger para generar información detallada, mensajes y códigos de error relacionados con su función.

```
# lambda_function.rb

require 'logger'

def handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info('## ENVIRONMENT VARIABLES')
  logger.info(ENV.to_a)
  logger.info('## EVENT')
  logger.info(event)
  event.to_a
end
```

La salida de `logger` incluye el nivel de registro, la marca de tiempo y el ID de la solicitud.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO] 2020-01-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ##
ENVIRONMENT VARIABLES

[INFO] 2020-01-31T22:12:58.534Z 1c8df7d3-xmpl-46da-9778-518e6eca8125
  environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
  'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d',
  'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})

[INFO] 2020-01-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 ## EVENT
```



```
[INFO] 2020-01-31T22:12:58.535Z 1c8df7d3-xmpl-46da-9778-518e6eca8125 {'key':  
'value'}
```

```
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
```

```
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Duration: 2.75 ms Billed
```

```
Duration: 3 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms
```

```
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861
```

```
Sampled: true
```

# Instrumentación del código Ruby en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle rastrear, depurar y optimizar las aplicaciones de Lambda. Puede utilizar X-Ray para rastrear una solicitud a medida que esta recorre los recursos de la aplicación, desde el frontend de la API hasta el almacenamiento y la base de datos del backend. Solo con agregar la biblioteca SDK de X-Ray a la configuración de compilación, puede registrar los errores y la latencia de cualquier llamada que realice la función a un servicio de AWS.

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.



Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

## Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.

## 6. Seleccione Guardar.

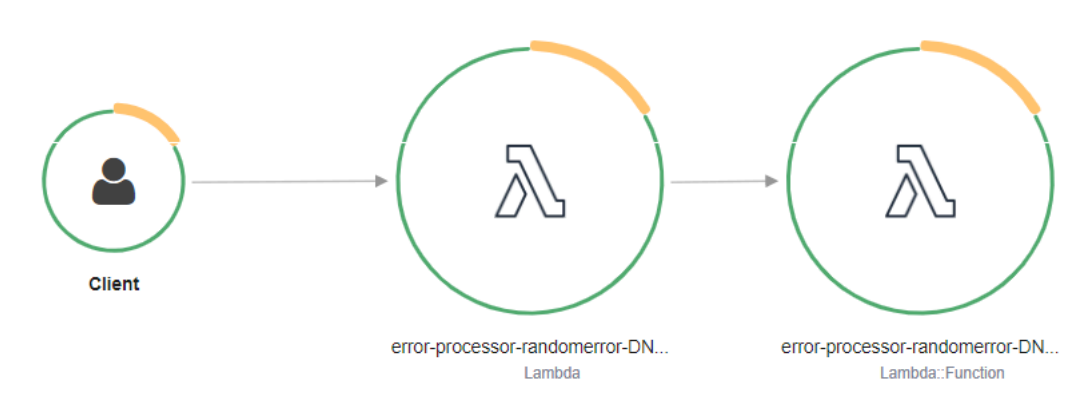
### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza cargos por almacenamiento y recuperación del seguimiento. Para más información, consulte [Precios de AWS X-Ray](#).

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

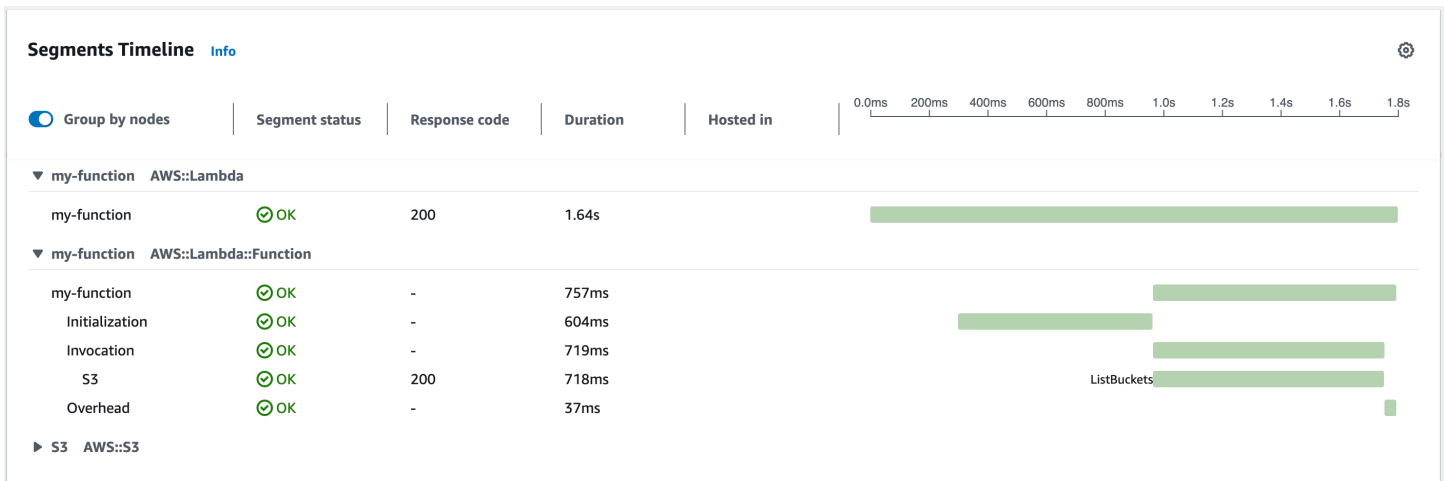
X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo, se muestra un seguimiento con estos dos segmentos. Ambos se denominan my-function, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el

segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

#### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.
- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.

- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

Puede instrumentar el código de controlador para registrar metadatos y rastrear llamadas descendentes. Para registrar detalles sobre las llamadas que su controlador realiza a otros recursos y servicios, use el X-Ray SDK para Ruby. Para obtener el SDK, agregue el paquete `aws-xray-sdk` a las dependencias de la aplicación.

Example [blank-ruby/function/Gemfile](#)

```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Para instrumentar clientes de AWS SDK, se requiere el módulo `aws-xray-sdk/lambda` después de crear un cliente en el código de inicialización.

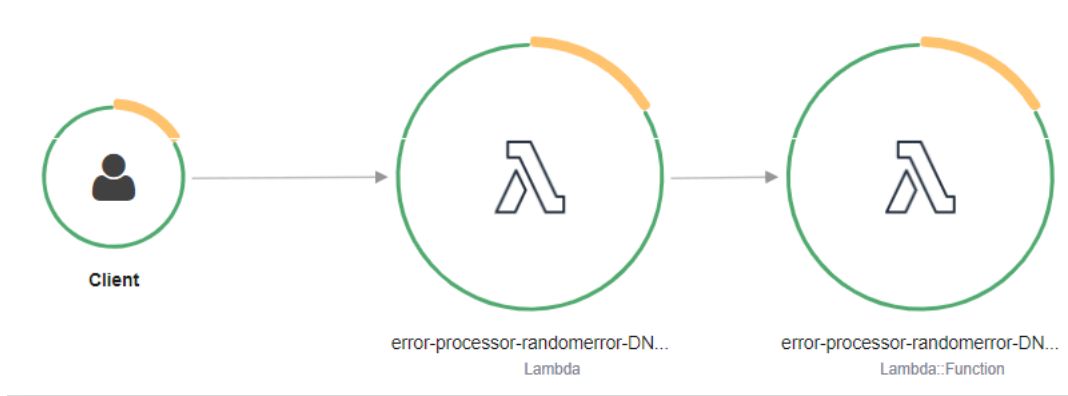
Example [blank-ruby/function/lambda\\_function.rb](#): seguimiento de un cliente SDK de AWS.

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

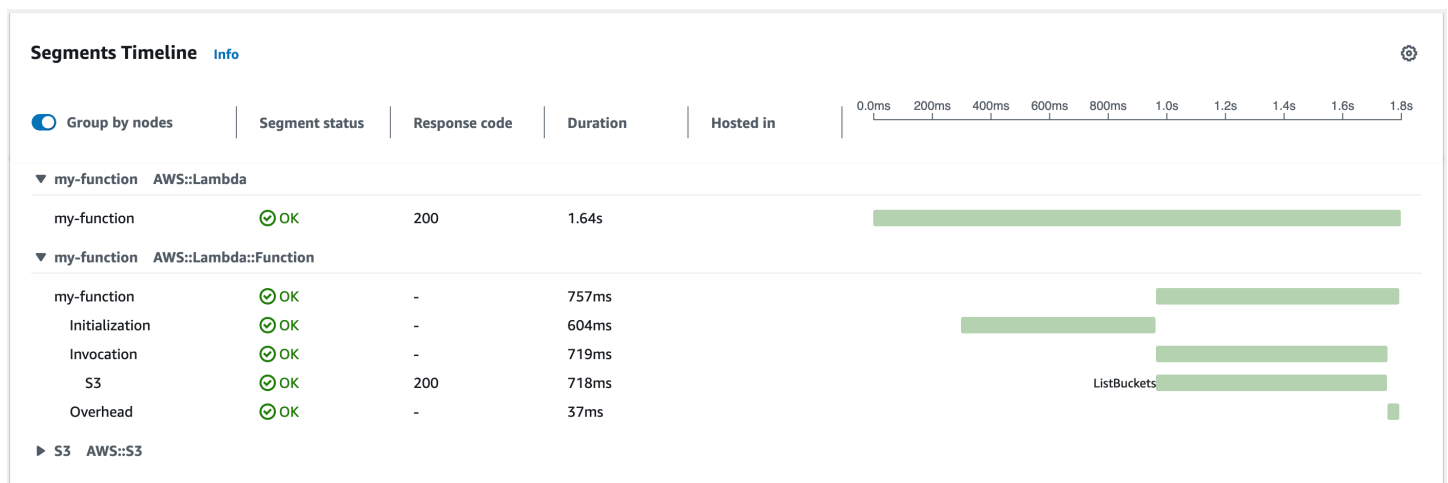
require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo, se muestra un seguimiento con estos dos segmentos. Ambos se denominan my-function, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

**Note**

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de

registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.
- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte [El X-Ray SDK para Ruby](#) en la Guía para desarrolladores de AWS X-Ray.

## Secciones

- [Habilitación del seguimiento activo con la API de Lambda](#)
- [Habilitación del seguimiento activo con AWS CloudFormation](#)
- [Almacenamiento de dependencias en tiempo de ejecución en una capa](#)

## Habilitación del seguimiento activo con la API de Lambda

Para administrar la configuración de seguimiento con la AWS CLI o el AWS SDK, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Habilitación del seguimiento activo con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:  
  function:
```



```
Type: AWS::Serverless::Function
Properties:
  Tracing: Active
  ...
```

## Almacenamiento de dependencias en tiempo de ejecución en una capa

Si utiliza el X-Ray SDK para instrumentar el código de las funciones de los clientes del SDK de AWS, el paquete de implementación puede llegar a ser bastante grande. Para evitar que se carguen dependencias en tiempo de ejecución cada vez que se actualice el código de las funciones, empaquete el X-Ray SDK en una [capa de Lambda](#).

En el ejemplo siguiente, se muestra un recurso de `AWS::Serverless::LayerVersion` que almacena el X-Ray SDK para Ruby.

Example [template.yml](#): capa de dependencias.

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-ruby-lib
      Description: Dependencies for the blank-ruby sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - ruby2.5
```

Con esta configuración, solo se actualiza la capa de la biblioteca si se modifican las dependencias del tiempo de ejecución. Dado que el paquete de implementación de la función contiene únicamente el código, esto puede ayudar a reducir los tiempos de carga.

Para crear una capa de dependencias, es necesario realizar cambios en la compilación para generar el archivo de capas antes de la implementación. Para ver un ejemplo de trabajo, consulte la aplicación de ejemplo [blank-ruby](#).

## Creación de funciones de Lambda con Java

Puede ejecutar código Java en AWS Lambda. Lambda proporciona [tiempos de ejecución](#) para Java que ejecutan código para procesar eventos. El código se ejecuta en un entorno de Amazon Linux que incluye las credenciales de AWS de un rol de AWS Identity and Access Management (IAM) administrado por usted.

Lambda admite los siguientes tiempos de ejecución de Java.

### Java

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloquear Actualizar función
Java 21	java21	Amazon Linux 2023			
Java 17	java17	Amazon Linux 2			
Java 11	java11	Amazon Linux 2			
Java 8	java8.a12	Amazon Linux 2			

Lambda proporciona las siguientes bibliotecas de funciones de Java:

- [com.amazonaws:aws-lambda-java-core](#) (obligatoria): define interfaces de métodos de controlador y el objeto contextual que el entorno de ejecución pasa al controlador. Si define sus propios tipos de entrada, esta es la única biblioteca que necesita.
- [com.amazonaws:aws-lambda-java-events](#): tipos de entrada de eventos procedentes de servicios que invocan funciones de Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): una biblioteca de appender de Log4j 2 para Apache que puede usar para agregar el ID solicitado en la invocación actual a los [registros de funciones](#).
- [SDK de AWS para Java 2.0](#): el SDK oficial de AWS para el lenguaje de programación Java.

**⚠ Important**

No utilice componentes privados (por ejemplo, campos, métodos o clases) de la API de JDK. Los componentes de la API que no son públicos pueden cambiar o eliminarse en cualquier actualización, y la aplicación dejará de funcionar.

Para crear una función Java

1. Abra la [consola de Lambda](#).
2. Elija Crear función.
3. Configure los siguientes ajustes:
  - En Nombre de la función: ingrese el nombre de la función.
  - En Tiempo de ejecución, elija Java 21.
4. Elija Crear función.
5. Para configurar un evento de prueba, seleccione Prueba.
6. En Nombre del evento, escriba **test**.
7. Elija Guardar cambios.
8. Elija Test (Probar) para invocar la función.

La consola crea una función de Lambda con una clase de controlador denominada `Hello`. Dado que Java es un lenguaje compilado, no puede ver ni editar el código fuente en la consola de Lambda, pero puede modificar su configuración, invocarla y configurar disparadores.

**📘 Note**

Para comenzar con el desarrollo de aplicaciones en su entorno local, implemente una de las [aplicaciones de ejemplo](#) disponibles en el repositorio de GitHub de esta guía.

La clase `Hello` cuenta con una función denominada `handleRequest` que toma un objeto de evento y un objeto `context`. Esta es la [función de controlador](#) a la que llama Lambda cuando se invoca la función. El tiempo de ejecución de la función de Java obtiene los eventos de invocación de Lambda y se los pasa al controlador. En la configuración de función, el valor de controlador es `example.Hello::handleRequest`.

Para actualizar el código de la función, cree un paquete de implementación, que es un archivo .zip que contiene el código de la función. A medida que avanza su función de desarrollo, querrá almacenar su código de función en el control del código fuente, agregar bibliotecas y automatizar las implementaciones. Comience [creando un paquete de implementación](#) y actualizando el código en la línea de comandos.

El tiempo de ejecución de la función pasa un objeto context al controlador, además del evento de invocación. El [objeto context](#) contiene información adicional acerca de la invocación, la función y el entorno de ejecución. Hay más información disponible en las variables de entorno.

Su función de Lambda tiene un grupo de registros de Registros de CloudWatch. El tiempo de ejecución de la función envía detalles de cada invocación a Registros de CloudWatch. Se transmite cualquier [registro que su función genere](#) durante la invocación. Si su función devuelve un error, Lambda formatea el error y lo devuelve al invocador.

## Temas

- [Definir el controlador de las funciones de Lambda en Java](#)
- [Implementar funciones de Lambda Java con archivos de archivo .zip o JAR](#)
- [Implementar funciones Java Lambda con imágenes de contenedor](#)
- [Uso de capas para funciones de Lambda en Java](#)
- [Mejora del rendimiento de inicio con Lambda SnapStart](#)
- [Personalice el comportamiento de inicio del tiempo de ejecución de Java para funciones de Lambda](#)
- [Uso del objeto de contexto Lambda para recuperar información de funciones de Java](#)
- [Registro y supervisión de las funciones de Lambda de Java](#)
- [Instrumentación del código Java en AWS Lambda](#)
- [Aplicaciones de ejemplo de Java para AWS Lambda](#)

# Definir el controlador de las funciones de Lambda en Java

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

El repositorio de GitHub para esta guía contiene aplicaciones de ejemplo fáciles de implementar en las que se muestran diferentes tipos de controladores. Para obtener más información, consulte el [final de este tema](#).

## Secciones

- [Ejemplo de controlador: tiempos de ejecución de Java 17](#)
- [Ejemplo de controlador: tiempos de ejecución de Java 11 e inferiores](#)
- [Código de inicialización](#)
- [Elección de los tipos de entrada y salida](#)
- [Interfaces de controlador](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda](#)
- [Ejemplo de código del controlador](#)

## Ejemplo de controlador: tiempos de ejecución de Java 17

En el siguiente ejemplo de Java 17, una clase llamada `HandlerIntegerJava17` define un método de controlador denominado `handleRequest`. El método del controlador incluye las siguientes entradas:

- Un `IntegerRecord`, que es un [registro](#) de Java personalizado que representa los datos de eventos. En este ejemplo, definimos `IntegerRecord` de la siguiente manera:

```
record IntegerRecord(int x, int y, String message) {  
}
```

- Un [objeto de contexto](#), que proporciona métodos y propiedades que brindan información sobre la invocación, la función y el entorno de ejecución.

Supongamos que queremos escribir una función que registre el `message` de la entrada `IntegerRecord` y devuelva la suma de `x` y `y`. El siguiente es el código de la función:

## Example [HandlerIntegerJava17.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

// Handler value: example.HandlerInteger
public class HandlerIntegerJava17 implements RequestHandler<IntegerRecord, Integer>{

    @Override
    /*
     * Takes in an InputRecord, which contains two integers and a String.
     * Logs the String, then returns the sum of the two Integers.
     */
    public Integer handleRequest(IntegerRecord event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        logger.log("String found: " + event.message());
        return event.x() + event.y();
    }
}

record IntegerRecord(int x, int y, String message) {
}
```

Usted especifica qué método desea que invoque Lambda al establecer el parámetro del controlador en la configuración de su función. Puede expresar el controlador en los siguientes formatos:

- ***package.Class::method***: formato completo. Por ejemplo:  
example.Handler::handleRequest.
- ***package.Class***: formato abreviado para las clases que implementan una interfaz de [controlador](#).  
Por ejemplo: example.Handler.

Cuando Lambda invoca su controlador, el [tiempo de ejecución](#) de Lambda recibe un evento como una cadena con formato JSON y lo convierte en un objeto. En el ejemplo anterior, un evento de ejemplo podría verse así:

## Example [evento.json](#)

```
{
  "x": 1,
  "y": 20,
  "message": "Hello World!"
}
```

Puede guardar este archivo y probar la función de forma local con el siguiente comando de AWS Command Line Interface (CLI):

```
aws lambda invoke --function-name function_name --payload file:///event.json out.json
```

## Ejemplo de controlador: tiempos de ejecución de Java 11 e inferiores

Lambda solo admite registros de Java 17 y tiempos de ejecución posteriores. En todos los entornos de ejecución de Java, puede usar una clase para representar los datos de eventos. En el siguiente ejemplo, se toma una lista de números enteros y un objeto de contexto como entrada y se devuelve la suma de todos los números enteros de la lista.

### Example [Handler.java](#)

En el siguiente ejemplo, la clase `Handler` define un método de controlador llamado `handleRequest`. El método `handler` toma un evento y un objeto contextual como entrada y devuelve una cadena.

### Example [HandlerList.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.List;

// Handler value: example.HandlerList
public class HandlerList implements RequestHandler<List<Integer>, Integer>{

    @Override
    /*
```

```
* Takes a list of Integers and returns its sum.
*/
public Integer handleRequest(List<Integer> event, Context context)
{
    LambdaLogger logger = context.getLogger();
    logger.log("EVENT TYPE: " + event.getClass().toString());
    return event.stream().mapToInt(Integer::intValue).sum();
}
}
```

Para ver más ejemplos, consulte [Ejemplo de código de controlador](#).

## Código de inicialización

Lambda ejecuta su código estático y el constructor de clases durante la [fase de inicialización](#) antes de invocar la función por primera vez. Los recursos que se crean durante la inicialización permanecen en la memoria entre las invocaciones y el controlador puede reutilizarlos miles de veces. Por lo tanto, puede agregar un [código de inicialización](#) fuera de su método de controlador principal para ahorrar tiempo de cómputo y reutilizar recursos en varias invocaciones.

En el siguiente ejemplo, el código de inicialización del cliente está fuera del método del controlador principal. El tiempo de ejecución inicializa el cliente antes de que la función publique su primer evento. Los eventos posteriores son mucho más rápidos porque Lambda no necesita volver a inicializar el cliente.

Example [Handler.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.Map;

import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.GetAccountSettingsResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String> {
```



```
private static final LambdaClient lambdaClient = LambdaClient.builder().build();

@Override
public String handleRequest(Map<String,String> event, Context context) {

    LambdaLogger logger = context.getLogger();
    logger.log("Handler invoked");

    GetAccountSettingsResponse response = null;
    try {
        response = lambdaClient.getAccountSettings();
    } catch(LambdaException e) {
        logger.log(e.getMessage());
    }
    return response != null ? "Total code size for your account is " +
response.accountLimit().totalCodeSize() + " bytes" : "Error";
}
}
```

## Elección de los tipos de entrada y salida

Especifique el tipo de objeto al que se asigna el evento en la firma del método del controlador. En el ejemplo anterior, el entorno de ejecución de Java deserializa el evento en un tipo que implementa la interfaz `Map<String, String>`. Los mapas de cadena a cadena funcionan para eventos planos como los siguientes:

Example [Event.json](#): datos del tiempo

```
{
  "temperatureK": 281,
  "windKmh": -3,
  "humidityPct": 0.55,
  "pressureHPa": 1020
}
```

Sin embargo, el valor de cada campo debe ser una cadena o un número. Si el evento contiene un campo que tiene un objeto como valor, el entorno de ejecución no puede deserializarlo y devuelve un error.

Elija un tipo de entrada que funcione con los datos del evento que procesa la función. Puede utilizar un tipo básico, un tipo genérico o un tipo bien definido.

## Tipos de entrada

- `Integer`, `Long`, `Double`, etc.: - El evento es un número sin formato adicional; por ejemplo, `3.5`. El entorno de ejecución convierte el valor en un objeto del tipo especificado.
- `String`: el evento es una cadena JSON, incluidas las comillas; por ejemplo, `"My string."`. El tiempo de ejecución convierte el valor (sin comillas) en un objeto `String`.
- `Type`, `Map<String, Type>`, etc.: - El evento es un objeto JSON. El entorno de ejecución lo deserializa en un objeto del tipo especificado o una interfaz.
- `List<Integer>`, `List<String>`, `List<Object>`, etc.: - El evento es una matriz JSON. El entorno de ejecución lo deserializa en un objeto del tipo especificado o una interfaz.
- `InputStream`: el evento es cualquier tipo JSON. El entorno de ejecución pasa una secuencia de bytes del documento al controlador sin modificaciones. Usted deserializa la entrada y escribe la salida en una secuencia de salida.
- Tipo de biblioteca: en el caso de los eventos enviados por los servicios de AWS, utilice los tipos de la biblioteca [aws-lambda-java-events](https://aws.amazon.com/blogs/compute/using-lambda-events/).

Si define su propio tipo de entrada, debería ser un objeto Java estándar (POJO) mutable y deserializable, con un constructor predeterminado y unas propiedades predeterminadas para cada campo del evento. Las claves del evento que no se asignan a una propiedad, así como las propiedades que no están incluidas en el evento, se eliminan sin errores.

El tipo de salida puede ser un objeto o `void`. El entorno de ejecución serializa los valores devueltos en texto. Si la salida es un objeto con campos, el entorno de ejecución lo serializa en un documento JSON. Si se trata de un tipo que encapsula un valor primitivo, el tiempo de ejecución devuelve una representación de texto de ese valor.

## Interfaces de controlador

La biblioteca [aws-lambda-java-core](https://aws.amazon.com/blogs/compute/using-lambda-events/) define dos interfaces para los métodos de controlador. Utilice las interfaces proporcionadas para simplificar la configuración del controlador y validar la firma del método del controlador en tiempo de compilación.

- [com.amazonaws.services.lambda.runtime.RequestHandler](https://aws.amazon.com/blogs/compute/using-lambda-events/)
- [com.amazonaws.services.lambda.runtime.RequestStreamHandler](https://aws.amazon.com/blogs/compute/using-lambda-events/)

La interfaz `RequestHandler` es un tipo genérico que toma dos parámetros: el tipo de entrada y el tipo de salida. Los dos tipos deben ser objetos. Cuando se utiliza esta interfaz, el entorno de ejecución de Java deserializa el evento en un objeto con el tipo de entrada y serializa la salida en texto. Utilice esta interfaz si la serialización integrada funciona con los tipos de entrada y salida.

Example [Handler.java](#): interfaz del controlador

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, String>{
    @Override
    public String handleRequest(Map<String,String> event, Context context)
```

Para usar su propia serialización, implemente la interfaz `RequestStreamHandler`. Con esta interfaz, Lambda pasa al controlador un flujo de entrada y otro flujo de salida. El controlador lee los bytes de la secuencia de entrada, escribe en la secuencia de salida y devuelve void.

En el ejemplo siguiente de Java 21 se muestra cómo puede utilizar una función de Lambda para procesar pedidos. En el ejemplo se utilizan tipos de lectores y escritores almacenados en el búfer para trabajar con las transmisiones de entrada y salida, y muestra cómo puede definir registros Java personalizados para utilizarlos en su función.

Example [HandlerStream.java](#)

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.List;

public class HandlerStream implements RequestStreamHandler {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    @Override
    public void handleRequest(InputStream input, OutputStream output, Context context)
        throws IOException {
        Order order = objectMapper.readValue(input, Order.class);
```

```
        processOrder(order);
        OrderAccepted orderAccepted = new OrderAccepted(order.orderId);

        objectMapper.writeValue(output, orderAccepted);
    }

    private void processOrder(Order order) {
        // business logic
    }

    public record Order(@JsonProperty("orderId") String orderId, @JsonProperty("items")
List<Item> items) { }

    public record Item(@JsonProperty("name") String name, @JsonProperty("quantity")
Integer quantity) { }

    public record OrderAccepted(@JsonProperty("orderId") String orderId) { }
}
```

## Prácticas recomendadas de codificación para las funciones de Lambda

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad.
- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#). Por ejemplo, es preferible utilizar los marcos de trabajo de inserción de dependencias (IoC) de Java más sencillos, como [Dagger](#) o [Guice](#), que los más complejos, como [Spring Framework](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación. En las funciones creadas en Java, evite

cargar toda la biblioteca de SDK de AWS como parte del paquete de implementación. En lugar de ello, cree dependencias selectivas de los módulos que seleccionen los componentes del SDK que necesita (por ejemplo, DynamoDB, módulos del SDK de Amazon S3 y [bibliotecas básicas de Lambda](#)).

- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.

- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).
- Evite utilizar la caché de DNS de Java. Las funciones de Lambda ya almacenan en caché las respuestas de DNS. Si utiliza otra caché de DNS, es posible que se agoten los tiempos de espera de conexión.

La clase `java.util.logging.Logger` puede habilitar indirectamente la caché de DNS de la JVM. Para anular la configuración predeterminada, defina [networkaddress.cache.ttl](#) en 0 antes de inicializar `logger`. Ejemplo:

```
public class MyHandler {
    // first set TTL property
    static{
        java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
    }
    // then instantiate logger
    var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

Para evitar errores de `UnknownHostException`, se recomienda establecer `networkaddress.cache.negative.ttl` en 0. Puede establecer esta propiedad para una función de Lambda con la variable de entorno `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

Al deshabilitar la caché de DNS de la JVM, no se deshabilita la caché de DNS administrada de Lambda.

- Reduzca el tiempo que tarda Lambda en desempaquetar los paquetes de implementación escritos en Java colocando los archivos `.jar` de dependencias en un directorio `/lib` independiente. Esto es más rápido que colocar todo el código de la función en un único archivo `jar` con un gran número de archivos `.class`. Para obtener instrucciones, consulte [Implementar funciones de Lambda Java con archivos de archivo .zip o JAR](#).

## Ejemplo de código del controlador

El repositorio de GitHub para esta guía contiene aplicaciones de ejemplo en las que se muestra el uso de diferentes tipos de controladores e interfaces. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación y la limpieza, una plantilla de AWS SAM y recursos de soporte.

### Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

Las aplicaciones `java-events` y `s3-java` toman un evento de un servicio de AWS como entrada y devuelven una cadena. La aplicación `java-basic` contiene varios tipos de controladores:

- [Handler.java](#): toma `Map<String, String>` como entrada.
- [HandlerInteger.java](#): toma `Integer` como entrada.
- [HandlerList.java](#): toma `List<Integer>` como entrada.
- [HandlerStream.java](#): toma `InputStream` y `OutputStream` como entrada.
- [HandlerString.java](#): toma `String` como entrada.
- [HandlerWeatherData.java](#): toma un tipo personalizado como entrada.

Para probar diferentes tipos de controlador, solo tiene que cambiar el valor del controlador en la plantilla de AWS SAM. Para obtener instrucciones detalladas, consulte el archivo readme (léame) de la aplicación de ejemplo.



# Implementar funciones de Lambda Java con archivos de archivo .zip o JAR

El código de la función AWS Lambda se compone de scripts o programas compilados y sus dependencias. Utiliza un paquete de implementación para implementar su código de función en Lambda. Lambda admite dos tipos de paquetes de implementación: imágenes de contenedor y archivos .zip.

Esta página describe cómo crear un archivo.zip o archivo Jar en su paquete de implementación y luego usar el archivo para implementar el código de función a AWS Lambda con el AWS Command Line Interface (AWS CLI).

## Secciones

- [Requisitos previos](#)
- [Herramientas y bibliotecas](#)
- [Compilación de un paquete de implementación con Gradle](#)
- [Creación de una capa de Java para las dependencias](#)
- [Compilación de un paquete de implementación con Maven](#)
- [Carga de un paquete de despliegue con la consola de Lambda](#)
- [Carga de un paquete de despliegue con la AWS CLI](#)
- [Carga de un paquete de implementación con AWS SAM](#)

## Requisitos previos

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

## Herramientas y bibliotecas

Lambda proporciona las siguientes bibliotecas de funciones de Java:

- [com.amazonaws:aws-lambda-java-core](#) (obligatoria): define interfaces de métodos de controlador y el objeto contextual que el entorno de ejecución pasa al controlador. Si define sus propios tipos de entrada, esta es la única biblioteca que necesita.
- [com.amazonaws:aws-lambda-java-events](#): tipos de entrada de eventos procedentes de servicios que invocan funciones de Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): una biblioteca de appender de Log4j 2 para Apache que puede usar para agregar el ID solicitado en la invocación actual a los [registros de funciones](#).
- [AWS SDK para Java 2.0](#): el SDK oficial de AWS para el lenguaje de programación Java.

Estas bibliotecas están disponibles en el [repositorio central de Maven](#). Agréguelas a la definición de la compilación de la siguiente manera:

## Gradle

```
dependencies {  
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'  
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'  
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'  
}
```

## Maven

```
<dependencies>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-core</artifactId>  
    <version>1.2.2</version>  
  </dependency>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-events</artifactId>  
    <version>3.11.1</version>  
  </dependency>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-lambda-java-log4j2</artifactId>  
    <version>1.5.1</version>  
  </dependency>  
</dependencies>
```

Para crear un paquete de implementación, compile el código de función y las dependencias en un único archivo .zip o Java Archive (JAR). En el caso de Gradle, [use el tipo de compilación Zip](#). En el caso de Apache Maven, [use el complemento Maven Shade](#). Cargue el paquete de despliegue a través de la consola de Lambda, la API de Lambda o AWS Serverless Application Model (AWS SAM).

### Note

Para que el tamaño del paquete de implementación sea reducido, empaquete las dependencias de la función en capas. Las capas le permiten administrar las dependencias de forma independiente, pueden utilizarlas varias funciones y pueden compartirse con otras cuentas. Para obtener más información, consulte [Capas de Lambda](#).

## Compilación de un paquete de implementación con Gradle

Utilice el tipo de compilación Zip para crear un paquete de despliegue con el código de función y las dependencias de Gradle. Este es un ejemplo de un [archivo build.gradle de muestra completo](#):

Example build.gradle: tarea de compilación

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
}
```

Esta configuración de compilación produce un paquete de implementación en el directorio build/distributions. Dentro de la instrucción into('lib'), la tarea jar crea un archivo JAR que contiene las clases principales en una carpeta denominada lib. A continuación, la tarea configurations.runtimeClassPath copia las bibliotecas de dependencias de la ruta de clases de la compilación en el mismo archivo lib.

Example build.gradle: dependencias

```
dependencies {
    ...
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
```

```
implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'  
implementation 'org.apache.logging.log4j:log4j-api:2.17.1'  
implementation 'org.apache.logging.log4j:log4j-core:2.17.1'  
runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.17.1'  
runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'  
...  
}
```

Lambda carga los archivos JAR en orden alfabético Unicode. Si hay varios archivos JAR en el directorio `lib` que contienen la misma clase, se utiliza el primero. Puede utilizar el siguiente script de shell para identificar las clases duplicadas.

Example test-zip.sh

```
mkdir -p expanded  
unzip path/to/my/function.zip -d expanded  
find ./expanded/lib -name '*.jar' | xargs -n1 zipinfo -1 | grep '.*.class' | sort |  
uniq -c | sort
```

## Creación de una capa de Java para las dependencias

### Note

El uso de capas con funciones en un lenguaje compilado, como Java, puede no ofrecer las mismas ventajas que con un lenguaje interpretado, como Python. Como Java es un lenguaje compilado, las funciones aún tienen que cargar de forma manual los ensamblajes compartidos en la memoria durante la fase de inicio, lo que puede aumentar los tiempos de arranque en frío. En su lugar, se recomienda incluir cualquier código compartido en el momento de la compilación para aprovechar las optimizaciones del compilador integradas.

En las instrucciones de esta sección, se muestra cómo incluir las dependencias en una capa. Para obtener instrucciones sobre cómo incluir las dependencias en el paquete de implementación, consulte [the section called “Compilación de un paquete de implementación con Gradle”](#) o [the section called “Compilación de un paquete de implementación con Maven”](#).

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio `/opt` de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable `PATH` ya incluye rutas de carpeta específicas en el directorio `/opt`. Para garantizar que la variable `PATH`

recoja el contenido de la capa, el archivo .zip de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

- java/lib (CLASSPATH)

Por ejemplo, la estructura del archivo .zip de la capa podría tener el siguiente aspecto:

```
jackson.zip
# java/lib/jackson-core-2.2.3.jar
```

Además, Lambda detecta de forma automática cualquier biblioteca en el directorio /opt/lib y todos los archivos binarios en el directorio /opt/bin. Para asegurarse de que Lambda encuentre el contenido de la capa de forma correcta, también puede crear una capa con la siguiente estructura:

```
custom-layer.zip
# lib
  | lib_1
  | lib_2
# bin
  | bin_1
  | bin_2
```

Después de empaquetar la capa, consulte [the section called “Creación y eliminación de capas”](#) y [the section called “Adición de capas”](#) para completar la configuración de la capa.

## Compilación de un paquete de implementación con Maven

Para crear un paquete de implementación con Maven, use el [complemento Maven Shade](#). El complemento crea un archivo JAR que contiene el código compilado de la función y todas sus dependencias.

Example pom.xml: configuración del complemento

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <createDependencyReducedPom>>false</createDependencyReducedPom>
  </configuration>
```

```
<executions>
  <execution>
    <phase>package</phase>
    <goals>
      <goal>shade</goal>
    </goals>
  </execution>
</executions>
</plugin>
```

Para crear el paquete de implementación, utilice el comando `mvn package`.

```
[INFO] Scanning for projects...
[INFO] -----< com.example:java-maven >-----
[INFO] Building java-maven-function 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
...
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ java-maven ---
[INFO] Building jar: target/java-maven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.2.2:shade (default) @ java-maven ---
[INFO] Including com.amazonaws:aws-lambda-java-core:jar:1.2.2 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:3.11.1 in the shaded jar.
[INFO] Including joda-time:joda-time:jar:2.6 in the shaded jar.
[INFO] Including com.google.code.gson:gson:jar:2.8.6 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing target/java-maven-1.0-SNAPSHOT.jar with target/java-maven-1.0-
SNAPSHOT-shaded.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.321 s
[INFO] Finished at: 2020-03-03T09:07:19Z
[INFO] -----
```

Este comando genera un archivo JAR en el directorio `target`.

#### Note

Si trabaja con un [JAR de varias versiones \(MRJAR\)](#), debe incluir el MRJAR (es decir, el JAR sombreado producido por el complemento Maven Shade) en el directorio `lib` y comprimirlo antes de subir el paquete de despliegue a Lambda. De lo contrario, es posible

que Lambda no descomprima correctamente el archivo JAR, lo que hará que se ignore el archivo MANIFEST.MF.

Si utiliza la biblioteca de appender (`aws-lambda-java-log4j2`), también debe configurar un transformador para el complemento Maven Shade. La biblioteca de transformadores combina versiones de un archivo de caché que aparecen tanto en la biblioteca de appender como en Log4j.

Example pom.xml: configuración del complemento con el appender Log4j 2

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <createDependencyReducedPom>>false</createDependencyReducedPom>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCacheFile
          </transformer>
        </transformers>
      </configuration>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>com.github.edwgiz</groupId>
      <artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
      <version>2.13.0</version>
    </dependency>
  </dependencies>
</plugin>
```

## Carga de un paquete de despliegue con la consola de Lambda

Para crear una nueva función, primero debe crearla en la consola y, a continuación, cargar el archivo .zip o JAR. Para actualizar una función existente, abra la página de la función y, a continuación, siga el mismo procedimiento para agregar el archivo .zip o JAR actualizado.

Si el archivo del paquete de despliegue tiene un tamaño inferior a los 50 MB, puede cargarlo directamente desde su equipo local. Para los archivos .zip o JAR de un tamaño mayor que 50 MB, es preciso cargar el paquete en un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS Management Console, consulte [Introducción a Amazon S3](#). Para cargar archivos mediante la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

### Note

No puede cambiar el [tipo de paquete de implementación](#) (.zip o imagen de contenedor) de una función existente. Por ejemplo, no se puede convertir una función de imagen de contenedor para utilizar un archivo .zip. Debe crear una nueva función.

Para crear una nueva función (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija Crear función.
2. Elija Crear desde cero.
3. En Información básica, haga lo siguiente:
  - a. En Nombre de la función, escriba el nombre de la función.
  - b. En Tiempo de ejecución, seleccione el tiempo de ejecución que desea utilizar.
  - c. (Opcional) Para Arquitectura, elija la arquitectura del conjunto de instrucciones para su función. La arquitectura predeterminada es x86\_64. Asegúrese de que el paquete de despliegue .zip para su función sea compatible con la arquitectura del conjunto de instrucciones que seleccione.
4. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o utilizar uno existente.
5. Elija Crear función. Lambda crea una función básica “Hola, mundo” mediante el tiempo de ejecución elegido.



Para cargar un archivo .zip o JAR desde su equipo local (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar el archivo .zip o JAR.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija archivo .zip o .jar.
5. Para cargar un archivo .zip o JAR, haga lo siguiente:
  - a. Seleccione Cargar y, a continuación, seleccione su archivo .zip o JAR en el selector de archivos.
  - b. Elija Abrir.
  - c. Seleccione Guardar.

Para cargar un archivo .zip o JAR desde un bucket de Amazon S3 (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar un nuevo archivo .zip o JAR.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija la ubicación de Amazon S3.
5. Pegue la URL del enlace de Amazon S3 de su archivo .zip y seleccione Guardar.

## Carga de un paquete de despliegue con la AWS CLI

Puede utilizar la [AWS CLI](#) para crear una nueva función o actualizar una existente mediante un archivo .zip o JAR. Utilice los comandos [create-function](#) y [update-function-code](#) para implementar el paquete .zip o JAR. Si el archivo tiene un tamaño inferior a los 50 MB, puede cargarlo desde su equipo de compilación local. Para archivos más grandes, es preciso cargar el paquete .zip o JAR desde un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

**Note**

Si carga su archivo .zip o JAR desde un bucket de Amazon S3 con la AWS CLI, el bucket debe estar ubicado en la misma Región de AWS que su función.

Para crear una nueva función mediante un archivo .zip o JAR con la AWS CLI, debe especificar lo siguiente:

- El nombre de la función (`--function-name`).
- El tiempo de ejecución de la función (`--runtime`).
- El nombre de recurso de Amazon (ARN) del [rol de ejecución](#) de la función (`--role`).
- El nombre del método de controlador en el código de la función (`--handler`).

También debe especificar la ubicación del archivo .zip o JAR. Si el archivo .zip o JAR se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, tal como se muestra en el siguiente comando de ejemplo.

```
aws lambda create-function --function-name myFunction \  
--runtime java21 --handler example.handler \  
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice la opción `--code`, como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `S3ObjectVersion` para los objetos con versiones.

```
aws lambda create-function --function-name myFunction \  
--runtime java21 --handler example.handler \  
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Para actualizar una función existente mediante la CLI, especifique el nombre de la función mediante el parámetro `--function-name`. También debe especificar la ubicación del archivo .zip que desea utilizar para actualizar el código de la función. Si el archivo .zip se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo .zip en un bucket de Amazon S3, utilice las opciones `--s3-bucket` y `--s3-key` tal como se muestra en el siguiente comando de ejemplo. Solo necesita usar el parámetro `--s3-object-version` para los objetos con versiones.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject  
Version
```

## Carga de un paquete de implementación con AWS SAM

Puede usar el AWS SAM para automatizar las implementaciones del código de función, la configuración y las dependencias. AWS SAM es una extensión de AWS CloudFormation que proporciona una sintaxis simplificada para definir aplicaciones sin servidor. En la siguiente plantilla de ejemplo, se define una función con un paquete de implementación en el directorio `build/distributions` que usa Gradle:

Example template.yml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: 'AWS::Serverless-2016-10-31'  
Description: An AWS Lambda application that calls the Lambda API.  
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: build/distributions/java-basic.zip  
      Handler: example.Handler  
      Runtime: java21  
      Description: Java function  
      MemorySize: 512  
      Timeout: 10  
      # Function's execution role  
    Policies:  
      - AWSLambdaBasicExecutionRole  
      - AWSLambda_ReadOnlyAccess  
      - AWSXrayWriteOnlyAccess  
      - AWSLambdaVPCLambdaAccessExecutionRole
```

Tracing: Active

Para crear la función, utilice los comandos `package` y `deploy`. Estos comandos son personalizaciones de la CLI de AWS CLI. Contienen otros comandos que van a cargar el paquete de implementación en Amazon S3, reescribir la plantilla con el URI del objeto y actualizar el código de la función.

El siguiente script de ejemplo ejecuta una compilación de Gradle y carga el paquete de implementación que crea. Crea una pila de AWS CloudFormation la primera vez que la ejecuta. Si la pila ya existe, el script la actualiza.

Example `deploy.sh`

```
#!/bin/bash
set -eo pipefail
aws cloudformation package --template-file template.yml --s3-bucket MY_BUCKET --output-template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name java-basic --capabilities CAPABILITY_NAMED_IAM
```

Para ver un ejemplo completo, consulte las siguientes aplicaciones:

Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.



# Implementar funciones Java Lambda con imágenes de contenedor

Hay tres formas de crear una imagen de contenedor para una función de Lambda en Java:

- [Uso de una imagen base de AWS para Java](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Java](#) en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para Java](#) en la imagen.

## Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.

## Temas

- [Imágenes base de AWS para Java](#)
- [Uso de una imagen base de AWS para Java](#)
- [Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución](#)

## Imágenes base de AWS para Java

AWS proporciona las siguientes imágenes base para Java:

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
21	Java 21	Amazon Linux 2023	<a href="#">Dockerfile para Java 21 en GitHub</a>	
17	Java 17	Amazon Linux 2	<a href="#">Dockerfile para Java 17 en GitHub</a>	
11	Java 11	Amazon Linux 2	<a href="#">Dockerfile para Java 11 en GitHub</a>	
8.al2	Java 8	Amazon Linux 2	<a href="#">Dockerfile para Java 8 en GitHub</a>	

Repositorio de Amazon ECR: [gallery.ecr.aws/lambda/java](https://gallery.ecr.aws/lambda/java)

Las imágenes base de Java 21 y versiones posteriores se basan en la [imagen de contenedor mínima de Amazon Linux 2023](#). Las imágenes base anteriores utilizan Amazon Linux 2. AL2023 ofrece varias ventajas con respecto a Amazon Linux 2, incluida una huella de implementación más reducida y versiones actualizadas de bibliotecas como `glibc`.

Las imágenes basadas en AL2023 utilizan `microdnf` (enlazadas simbólicamente como `dnf`) como administrador de paquetes en lugar de `yum`, que es el administrador de paquetes predeterminado en Amazon Linux 2. `microdnf` es una implementación independiente de `dnf`. Para obtener una lista de los paquetes que se incluyen en las imágenes basadas en AL2023, consulte las columnas de Minimal Container de [Comparing packages installed on Amazon Linux 2023 Container Images](#). Para obtener más información sobre las diferencias entre AL2023 y Amazon Linux 2, consulte [Introducing the Amazon Linux 2023 runtime for AWS Lambda](#) en el Blog de informática de AWS.

**Note**

Para ejecutar imágenes basadas en AL2023 de forma local, incluso con AWS Serverless Application Model (AWS SAM), debe usar Docker en la versión 20.10.10 o posterior.

## Uso de una imagen base de AWS para Java

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Java (por ejemplo, [Amazon Corretto](#))
- [Docker](#) (versión mínima 20.10.10 para las imágenes base de Java 21 y versiones posteriores)
- [Apache Maven](#) o [Gradle](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

### Creación de una imagen a partir de una imagen base

#### Maven

1. Ejecute el siguiente comando para crear un proyecto de Maven con el [arquetipo de Lambda](#). Se requieren los siguientes parámetros:
  - `service`: el cliente del Servicio de AWS que se va a utilizar en la función de Lambda. Para ver una lista de las fuentes disponibles, consulte [aws-sdk-java-v2/services](#) en GitHub.
  - `region`: la Región de AWS en la que desea crear la función de Lambda.
  - `groupId`: el espacio de nombres completo del paquete de la aplicación.
  - `artifactId`: el nombre del proyecto. Esto se convierte en el nombre del directorio del proyecto.

En Linux y macOS, ejecute este comando:

```
mvn -B archetype:generate \  
  -DarchetypeGroupId=software.amazon.awssdk \  
  -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \  
  -DgroupId=com.example.myapp \  
  -DartifactId=example
```



```
-DartifactId=myapp
```

En PowerShell, ejecute este comando:

```
mvn -B archetype:generate `
  "-DarchetypeGroupId=software.amazon.awssdk" `
  "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2"
  `
  "-DgroupId=com.example.myapp" `
  "-DartifactId=myapp"
```

El arquetipo de Maven para Lambda está preconfigurado para compilar con Java SE 8 e incluye una dependencia a AWS SDK for Java. Si crea su proyecto con un arquetipo diferente o mediante otro método, debe [configurar el compilador de Java para Maven](#) y [declarar el SDK como dependencia](#).

2. Abra el directorio *myapp/src/main/java/com/example/myapp* y busque el archivo `App.java`. Se trata del código de la función de Lambda. Puede utilizar el código de muestra proporcionado para realizar pruebas o sustituirlo por su propio código.
3. Regrese al directorio raíz del proyecto y, a continuación, cree un nuevo Dockerfile con la siguiente configuración:
  - Establezca la propiedad FROM en el [URI de la imagen base](#).
  - Establezca el argumento CMD para el controlador de la función de Lambda.

#### Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Maven layout
COPY target/classes ${LAMBDA_TASK_ROOT}
COPY target/dependency/* ${LAMBDA_TASK_ROOT}/lib/

# Set the CMD to your handler (could also be done as a parameter override
  outside of the Dockerfile)
CMD [ "com.example.myapp.App::handleRequest" ]
```

4. Compile el proyecto y recopile las dependencias del tiempo de ejecución.

```
mvn compile dependency:copy-dependencies -DincludeScope=runtime
```

5. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

## Gradle

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir example  
cd example
```

2. Ejecute el siguiente comando para que Gradle genere un nuevo proyecto de aplicación Java en el directorio `example` de su entorno. En Seleccionar DSL del script de compilación, elija 2: Groovy.

```
gradle init --type java-application
```

3. Abra el directorio `/example/app/src/main/java/example` y busque el archivo `App.java`. Se trata del código de la función de Lambda. Puede utilizar el siguiente código de muestra para realizar pruebas o sustituirlo por su propio código.

### Example App.java

```
package com.example;  
import com.amazonaws.services.lambda.runtime.Context;
```

```
import com.amazonaws.services.lambda.runtime.RequestHandler;
public class App implements RequestHandler<Object, String> {
    public String handleRequest(Object input, Context context) {
        return "Hello world!";
    }
}
```

- Abra el archivo `build.gradle`. Si utiliza el código de función de muestra del paso anterior, sustituya el contenido de `build.gradle` por lo siguiente. Si utiliza su propio código de función, modifique el archivo `build.gradle` según sea necesario.

#### Example build.gradle (Groovy DSL)

```
plugins {
    id 'java'
}
group 'com.example'
version '1.0-SNAPSHOT'
sourceCompatibility = 1.8
repositories {
    mavenCentral()
}
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
}
jar {
    manifest {
        attributes 'Main-Class': 'com.example.App'
    }
}
```

- El comando `gradle init` del paso 2 también generó un caso de prueba ficticio en el directorio `app/test`. Para los fines de este tutorial, evite la ejecución de las pruebas al eliminar el directorio de `/test`.
- Compilar el proyecto.

```
gradle build
```

- En el directorio raíz del proyecto (`/example`), cree un `Dockerfile` con la siguiente configuración:

- Establezca la propiedad FROM en el [URI de la imagen base](#).

- Utilice el comando COPY para copiar el código de la función y las dependencias del tiempo de ejecución a `{LAMBDA_TASK_ROOT}`, una [variable de entorno definido de Lambda](#).
- Establezca el argumento CMD para el controlador de la función de Lambda.

### Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Gradle layout
COPY app/build/classes/java/main ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override
  outside of the Dockerfile)
CMD [ "com.example.App::handleRequest" ]
```

8. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

### (Opcional) Prueba de la imagen de forma local

1. Inicie la imagen de Docker con el comando `docker run`. En este ejemplo, `docker-image` es el nombre de la imagen y `test` es la etiqueta.

```
docker run --platform linux/amd64 -p 9000:8080 docker-image:test
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

2. Desde una nueva ventana de terminal, publique un evento en el punto de conexión local.

## Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{"payload":"hello world!"}'
```

## PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

### 3. Obtenga el ID del contenedor.

```
docker ps
```

### 4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

#### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
```

```
"repository": {
  "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
  "registryId": "111122223333",
  "repositoryName": "hello-world",
  "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
  "createdAt": "2023-03-09T10:39:01+00:00",
  "imageTagMutability": "MUTABLE",
  "imageScanningConfiguration": {
    "scanOnPush": true
  },
  "encryptionConfiguration": {
    "encryptionType": "AES256"
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.

7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.



## Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución

Si usa una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir el cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución extiende el [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de la función.

Instale el cliente de interfaz de tiempo de ejecución para Java en su Dockerfile o como una dependencia en su proyecto. Por ejemplo, para instalar el cliente de interfaz de tiempo de ejecución mediante el administrador de paquetes de Maven, agregue los siguientes a su archivo pom.xml:

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
  <version>2.3.2</version>
</dependency>
```

Para obtener detalles sobre el paquete, consulte [AWS Lambda Java Runtime Interface Client](#) en Maven Central Repository. También puede ver el código fuente del cliente de interfaz de tiempo de ejecución de Java en el repositorio de GitHub [AWS Lambda Java Support Libraries](#).

En el siguiente ejemplo, se muestra cómo crear una imagen de contenedor para Java mediante una [imagen de Amazon Corretto](#). Amazon Corretto es una distribución sin costo, multiplataforma y lista para producción de Open Java Development Kit (OpenJDK). El proyecto Maven incluye el cliente de interfaz de tiempo de ejecución como una dependencia.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Java (por ejemplo, [Amazon Corretto](#))
- [Docker](#)
- [Apache Maven](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

### Creación de imágenes a partir de una imagen base alternativa

1. Cree un proyecto de Maven. Se requieren los siguientes parámetros:

- `groupId`: el espacio de nombres completo del paquete de la aplicación.
- `artifactId`: el nombre del proyecto. Esto se convierte en el nombre del directorio del proyecto.

## Linux/macOS

```
mvn -B archetype:generate \  
  -DarchetypeArtifactId=maven-archetype-quickstart \  
  -DgroupId=example \  
  -DartifactId=myapp \  
  -DinteractiveMode=false
```

## PowerShell

```
mvn -B archetype:generate \  
  -DarchetypeArtifactId=maven-archetype-quickstart \  
  -DgroupId=example \  
  -DartifactId=myapp \  
  -DinteractiveMode=false
```

2. Abra el directorio del proyecto.

```
cd myapp
```

3. Reemplace el contenido del archivo `pom.xml` por lo siguiente. Este archivo contiene el [aws-lambda-java-runtime-interface-client](#) como dependencia. Como opción, puede instalar el cliente de interfaz de tiempo de ejecución en el `Dockerfile`. Sin embargo, el enfoque más simple es incluir la biblioteca como una dependencia.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://  
www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/  
maven-v4_0_0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  <groupId>example</groupId>  
  <artifactId>hello-lambda</artifactId>  
  <packaging>jar</packaging>  
  <version>1.0-SNAPSHOT</version>  
  <name>hello-lambda</name>  
  <url>http://maven.apache.org</url>  
  <properties>
```

```
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>2.3.2</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.1.2</version>
      <executions>
        <execution>
          <id>copy-dependencies</id>
          <phase>package</phase>
          <goals>
            <goal>copy-dependencies</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</project>
```

4. Abra el directorio `myapp/src/main/java/com/example/myapp` y busque el archivo `App.java`. Se trata del código de la función de Lambda. Reemplace el código con lo siguiente.

#### Example controlador de funciones

```
package example;

public class App {
    public static String sayHello() {
        return "Hello world!";
    }
}
```

5. El comando `mvn -B archetype:generate` del paso 1 también generó un caso de prueba ficticio en el directorio `src/test`. Para los fines de este tutorial, evite la ejecución de las pruebas y elimine todo este directorio de `/test` generado.
6. Regrese al directorio raíz del proyecto y, a continuación, crea un nuevo Dockerfile. El siguiente Dockerfile de ejemplo usa una [imagen de Amazon Corretto](#). Amazon Corretto es una distribución sin costo, multiplataforma y lista para producción de OpenJDK.
  - Establezca la propiedad `FROM` en el URI de la imagen base.
  - Configure `ENTRYPOINT` como el módulo que desea que el contenedor de Docker ejecute cuando se inicie. En este caso, el módulo es el cliente de interfaz de tiempo de ejecución.
  - Establezca el argumento `CMD` para el controlador de la función de Lambda.

### Example Dockerfile

```
FROM public.ecr.aws/amazoncorretto/amazoncorretto:21 as base

# Configure the build environment
FROM base as build
RUN yum install -y maven
WORKDIR /src

# Cache and copy dependencies
ADD pom.xml .
RUN mvn dependency:go-offline dependency:copy-dependencies

# Compile the function
ADD . .
RUN mvn package

# Copy the function artifact and dependencies onto a clean base
FROM base
WORKDIR /function

COPY --from=build /src/target/dependency/*.jar ./
COPY --from=build /src/target/*.jar ./

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/bin/java", "-cp", ".*",
"com.amazonaws.services.lambda.runtime.api.client.AWSLambda" ]
# Pass the name of the function handler as an argument to the runtime
```

```
CMD [ "example.App::sayHello" ]
```

7. Cree la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

(Opcional) Prueba de la imagen de forma local

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. Puede [crear el emulador en su imagen](#) o usar el procedimiento siguiente para instalarlo en su equipo local.

Para instalar y ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Desde el directorio del proyecto, ejecute el siguiente comando para descargar el emulador de interfaz de tiempo de ejecución (arquitectura x86-64) de GitHub e instalarlo en su equipo local.

Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Para instalar el emulador arm64, reemplace la URL del repositorio de GitHub en el comando anterior por lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

## PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Para instalar el emulador arm64, reemplace el `$downloadLink` con lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:

- `docker-image` es el nombre de la imagen y `test` es la etiqueta.
- `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello` es el ENTRYPOINT seguido del CMD de su Dockerfile.

## Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    docker-image:test \
        /usr/bin/java -cp './*'
        com.amazonaws.services.lambda.runtime.api.client.AWSLambda
        example.App::sayHello
```

## PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
```

```
docker-image:test `
  /usr/bin/java -cp './*'
  com.amazonaws.services.lambda.runtime.api.client.AWSLambda
  example.App::sayHello
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

3. Publique un evento en el punto de conexión local.

### Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
'{"payload":"hello world!"}'
```

### PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType "application/json"
```

4. Obtenga el ID del contenedor.

```
docker ps
```

5. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace 3766c4ab331c por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace 111122223333 por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```



**Note**

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \  
  --function-name hello-world \  
  --package-type Image \  
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \  
  --role arn:aws:iam::111122223333:role/lambda-ex
```

#### Note

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

8. Invoque la función.

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

9. Para ver la salida de la función, compruebe el archivo `response.json`.

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

# Uso de capas para funciones de Lambda en Java

Una [capa de Lambda](#) es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración. La creación de una capa implica tres pasos generales:

1. Empaquete el contenido de su capa. Esto significa crear un archivo de archivo. zip que contenga las dependencias que desea usar en sus funciones.
2. Cree la capa en Lambda.
3. Agregue la capa a sus funciones.

Este tema contiene los pasos y las instrucciones sobre cómo empaquetar y crear correctamente una capa de Lambda en Java con dependencias de bibliotecas externas.

## Temas

- [Requisitos previos](#)
- [Compatibilidad de capas de Java con Amazon Linux](#)
- [Rutas de capa para tiempos de ejecución de Java](#)
- [Empaquetado del contenido de la capa](#)
- [Creación de la capa](#)
- [Agregue la capa a la función](#)

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [Java 21](#)
- [Apache Maven versión 3.8.6 o posterior](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

### Note

Asegúrese de que la versión de Java a la que hace referencia Maven sea la misma que la versión de Java de la función que pretende implementar. Por ejemplo, para una función de Java 21, el comando `mvn -v` debería incluir la versión 21 de Java en el resultado:

```
Apache Maven 3.8.6
...
Java version: 21.0.2, vendor: Oracle Corporation, runtime: /Library/Java/
JavaVirtualMachines/jdk-21.jdk/Contents/Home
...
```

A lo largo de este tema, haremos referencia a la aplicación de muestra [layer-java](#) en el repositorio de GitHub [awsdocs](#). Esta aplicación contiene scripts que descargan las dependencias y generan las capas. La aplicación también contiene las funciones correspondientes que utilizan las dependencias de las capas. Tras crear una capa, puede implementar e invocar la función correspondiente para comprobar que todo funciona correctamente. Debido a que utiliza el tiempo de ejecución de Java 21 para las funciones, las capas también deben ser compatibles con Java 21.

La aplicación `layer-java` de ejemplo contiene un único ejemplo en dos subdirectorios. El directorio `layer` contiene un archivo `pom.xml` que define las dependencias de la capa, así como los scripts para generar la capa. El directorio `function` contiene una función de ejemplo que ayuda a comprobar el funcionamiento de la capa. Este tutorial explica cómo crear y empaquetar esta capa.

## Compatibilidad de capas de Java con Amazon Linux

El primer paso para crear una capa consiste en agrupar todo el contenido de la capa en un archivo `.zip`. Dado que las funciones de Lambda se ejecutan en [Amazon Linux](#), el contenido de la capa debe poder compilarse y crearse en un entorno de Linux.

El código Java está diseñado para ser independiente de la plataforma, por lo que puede empaquetar las capas en su máquina local aunque no utilice un entorno Linux. Tras cargar la capa de Java en Lambda, seguirá siendo compatible con Amazon Linux.

## Rutas de capa para tiempos de ejecución de Java

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio `/opt` de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable `PATH` ya incluye rutas de carpeta específicas en el directorio `/opt`. Para garantizar que la variable `PATH` recoja el contenido de la capa, el archivo `.zip` de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

- `java/lib`

Por ejemplo, el archivo.zip de capa resultante que cree en este tutorial tiene la siguiente estructura de directorios:

```
layer_content.zip
# java
  # lib
    # layer-java-layer-1.0-SNAPSHOT.jar
```

El archivo JAR `layer-java-layer-1.0-SNAPSHOT.jar` (un uber-jar que contiene todas las dependencias necesarias) está ubicado correctamente en el directorio `java/lib`. Esto garantiza que Lambda pueda localizar la biblioteca durante las invocaciones de funciones.

## Empaquetado del contenido de la capa

En este ejemplo, empaquetará las dos bibliotecas Java siguientes en un único archivo JAR:

- [aws-lambda-java-core](#): conjunto mínimo de definiciones de interfaz para trabajar con Java en AWS Lambda
- [Jackson](#): un popular conjunto de herramientas de procesamiento de datos, especialmente para trabajar con JSON.

Siga los pasos que se indican a continuación para instalar y empaquetar el contenido de la capa.

Para instalar y empaquetar el contenido de la capa

1. Clone el [repositorio de aws-lambda-developer-guide de GitHub](#), que contiene el código de muestra que necesita en el directorio `sample-apps/layer-java`.

```
git clone https://github.com/awsdocs/aws-lambda-developer-guide.git
```

2. Navegue hasta el directorio `layer` de la aplicación de ejemplo `layer-java`. Este directorio contiene los scripts que usa para crear y empaquetar la capa correctamente.

```
cd aws-lambda-developer-guide/sample-apps/layer-java/layer
```

3. Examine el archivo [pom.xml](#). En la sección `<dependencies>`, defina las dependencias que desea incluir en la capa, es decir, las bibliotecas `aws-lambda-java-core` y `jackson-databind`. Puede actualizar este archivo para incluir cualquier dependencia que desee incluir en su propia capa.

## Example pom.xml

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-core</artifactId>
    <version>1.2.3</version>
  </dependency>

  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.17.0</version>
  </dependency>
</dependencies>
```

### Note

La sección `<build>` de este archivo `pom.xml` contiene dos complementos. [maven-compiler-plugin](#) que compila el código fuente. [maven-shade-plugin](#) que empaqueta sus artefactos en un único uber-jar.

4. Asegúrese de tener los permisos para ejecutar ambos scripts.

```
chmod 744 1-install.sh && chmod 744 2-package.sh
```

5. Ejecute el script [1-install.sh](#) mediante el siguiente comando:

```
./1-install.sh
```

Este script ejecuta `mvn clean install` en el directorio actual. Esto crea el uber-jar con todas las dependencias necesarias en el directorio `target/`.

### Example 1-install.sh

```
mvn clean install
```

6. Ejecute el script [2-package.sh](#) mediante el siguiente comando:

```
./2-package.sh
```

Este script crea la estructura de directorios `java/lib` que necesita para empaquetar correctamente el contenido de la capa. A continuación, copia el `uber-jar` del directorio `/target` al directorio `java/lib` recién creado. Finalmente, el script comprime el contenido del directorio `java` en un archivo llamado `layer_content.zip`. Este es el archivo `.zip` para su capa. Puede descomprimir el archivo y comprobar que contiene la estructura de archivos correcta, como se muestra en la sección [the section called “Rutas de capa para tiempos de ejecución de Java”](#).

Example 2-package.sh

```
mkdir java
mkdir java/lib
cp -r target/layer-java-layer-1.0-SNAPSHOT.jar java/lib/
zip -r layer_content.zip java
```

## Creación de la capa

En esta sección, seleccione el archivo `layer_content.zip` que generó en la sección anterior y cárguelo como una capa de Lambda. Puede cargar una capa mediante la AWS Management Console o la API de Lambda en la AWS Command Line Interface (AWS CLI). Al cargar el archivo `.zip` de la capa, en el siguiente comando de AWS CLI [PublishLayerVersion](#), especifique `java21` como tiempo de ejecución compatible y `arm64` como arquitectura compatible.

```
aws lambda publish-layer-version --layer-name java-jackson-layer \
  --zip-file fileb://layer_content.zip \
  --compatible-runtimes java21 \
  --compatible-architectures "arm64"
```

En la respuesta, anote el `LayerVersionArn`, que tiene este aspecto `arn:aws:lambda:us-east-1:123456789012:layer:java-jackson-layer:1`. Necesitará este nombre de recurso de Amazon (ARN) en el siguiente paso de este tutorial cuando agregue la capa a la función.

## Agregue la capa a la función

En esta sección, despliega una función de Lambda de ejemplo que utiliza la biblioteca Jackson en su código de función y a continuación adjunta la capa. Para implementar la función, necesita un [the](#)



[section called “Rol de ejecución \(permisos para que las funciones accedan a otros recursos\)”](#). Si no dispone de un rol de ejecución existente, siga los pasos de la sección desplegable. De no ser así, vaya a la siguiente sección para implementar la función.

(Opcional) Cree un rol de ejecución

Para crear un rol de ejecución

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Trusted entity (Entidad de confianza).–Lambda:.
  - Permisos: AWSLambdaBasicExecutionRole.
  - Nombre de rol: **lambda-role**.

La política AWSLambdaBasicExecutionRole tiene permisos que la función necesita para escribir registros a Registros de CloudWatch.

Para implementar la función de Lambda

1. Vaya al directorio `function/`. Si se encuentra actualmente en el directorio `layer/`, ejecute el siguiente comando:

```
cd ../function
```

2. Revise el [código de la función](#). La función toma a `Map<String, String>` como entrada y usa Jackson para escribir la entrada como una cadena JSON antes de convertirla en un objeto Java `F1Car` predefinido. Por último, la función utiliza los campos del objeto `F1Car` para crear una cadena que devuelve la función.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.util.Map;
```

```
public class Handler {  
  
    public String handleRequest(Map<String, String> input, Context context) throws  
    IOException {  
        // Parse the input JSON  
        ObjectMapper objectMapper = new ObjectMapper();  
        F1Car f1Car =  
objectMapper.readValue(objectMapper.writeValueAsString(input), F1Car.class);  
  
        StringBuilder finalString = new StringBuilder();  
        finalString.append(f1Car.getDriver());  
        finalString.append(" is a driver for team ");  
        finalString.append(f1Car.getTeam());  
        return finalString.toString();  
    }  
}
```

3. Cree el proyecto usando el siguiente comando de Maven:

```
mvn package
```

Este comando genera un archivo JAR en el directorio `target/` denominado `layer-java-function-1.0-SNAPSHOT.jar`.

4. Implemente la función. En el siguiente comando AWS CLI, sustituya el parámetro `--role` por el ARN del rol de ejecución:

```
aws lambda create-function --function-name java_function_with_layer \  
    --runtime java21 \  
    --architectures "arm64" \  
    --handler example.Handler::handleRequest \  
    --timeout 30 \  
    --role arn:aws:iam::123456789012:role/lambda-role \  
    --zip-file fileb://target/layer-java-function-1.0-SNAPSHOT.jar
```

(Opcional) Invocar su función sin adjuntar una capa

En este punto, si lo desea, puede intentar invocar su función antes de adjuntar la capa. Si lo intenta, aparecerá un `ClassNotFoundException` porque la función no puede hacer referencia al paquete `requests`. Utilice el comando AWS CLI para invocar la función:

```
aws lambda invoke --function-name java_function_with_layer \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "driver": "Max Verstappen", "team": "Red Bull" }' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "StatusCode": 200,  
  "FunctionError": "Unhandled",  
  "ExecutedVersion": "$LATEST"  
}
```

Para ver el error específico, abra el archivo de salida `response.json`. Debería ver un `ClassNotFoundException` con el siguiente mensaje de error:

```
"errorMessage":"com.fasterxml.jackson.databind.ObjectMapper","errorType":"java.lang.ClassNotFou
```

A continuación, adjunte la capa a la función. En el siguiente comando AWS CLI, sustituya el parámetro `--layers` por el ARN de la versión de capa que indicó anteriormente:

```
aws lambda update-function-configuration --function-name java_function_with_layer \  
  --cli-binary-format raw-in-base64-out \  
  --layers "arn:aws:lambda:us-east-1:123456789012:layer:java-jackson-layer:1"
```

Finalmente, intente invocar su función usando el siguiente comando AWS CLI:

```
aws lambda invoke --function-name java_function_with_layer \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "driver": "Max Verstappen", "team": "Red Bull" }' response.json
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"  
}
```

Esto indica que la función pudo usar la dependencia de Jackson para ejecutar la función correctamente. Puede comprobar que el archivo de salida `response.json` contiene la cadena devuelta correcta:

```
"Max Verstappen is a driver for team Red Bull"
```

### (Opcional) Eliminar sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

Para eliminar la capa de Lambda

1. Abra la página de [Capas](#) de la consola de Lambda.
2. Seleccione la capa que ha creado.
3. Elija Eliminar, luego, vuelva a elegir Eliminar.

Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y seleccione Delete (Eliminar).

# Mejora del rendimiento de inicio con Lambda SnapStart

Lambda SnapStart para Java puede mejorar hasta 10 veces el rendimiento de inicio en las aplicaciones con exigencias en materia de latencia, sin costo adicional y normalmente sin cambios en el código de la función. El factor que más contribuye a la latencia de inicio (a menudo denominado tiempo de inicio en frío) es el tiempo que Lambda dedica a inicializar la función, que incluye cargar el código de la función, iniciar el tiempo de ejecución e inicializar el código de la función.

Con SnapStart, Lambda inicializa la función al publicar una versión de la función. Lambda toma una instantánea de [Firecracker microVM](#) del estado de la memoria y el disco correspondientes al [entorno de ejecución](#) inicializado, cifra la instantánea y la almacena en caché para acceder a ella con baja latencia. Al invocar la versión de la función por primera vez y a medida que las invocaciones escalan verticalmente, Lambda reanuda los nuevos entornos de ejecución a partir de la instantánea almacenada en caché, en lugar de inicializarlos desde cero, lo que mejora la latencia de inicio.

## Important

Si sus aplicaciones dependen de la singularidad del estado, debe evaluar el código de su función y comprobar que es resistente a las operaciones de instantáneas. Para obtener más información, consulte [Control de la exclusividad con Lambda SnapStart](#).

## Temas

- [Características y limitaciones compatibles](#)
- [Regiones admitidas](#)
- [Consideraciones sobre compatibilidad](#)
- [Precios de SnapStart](#)
- [Comparación entre Lambda SnapStart y la simultaneidad aprovisionada](#)
- [Recursos adicionales de](#)
- [Activación y administración de Lambda SnapStart](#)
- [Control de la exclusividad con Lambda SnapStart](#)
- [Implemente el código antes o después de las instantáneas de la función de Lambda](#)
- [Monitoreo para Lambda SnapStart](#)
- [Modelo de seguridad para Lambda SnapStart](#)
- [Maximice el rendimiento de Lambda SnapStart](#)

## Características y limitaciones compatibles

SnapStart admite Java 11 y los [tiempos de ejecución administrados de Java](#) posteriores. No se admiten otros tiempos de ejecución administrados (como `nodejs20.x` y `python3.12`), [Tiempos de ejecución exclusivos del sistema operativo](#), ni [imágenes de contenedor](#)

SnapStart no admite la [simultaneidad aprovisionada](#), [Amazon Elastic File System \(Amazon EFS\)](#), ni almacenamiento efímero de más de 512 MB.

Para trabajar con SnapStart, puede utilizar la consola de Lambda, la AWS Command Line Interface (AWS CLI), la API de Lambda, AWS SDK for Java, el servicio AWS CloudFormation, AWS Serverless Application Model (AWS SAM) y AWS Cloud Development Kit (AWS CDK). Para obtener más información, consulte [Activación y administración de Lambda SnapStart](#).

### Note

Puede usar SnapStart solo en las [versiones de funciones publicadas](#) y en los [alias](#) que apunten a versiones. No puede usar SnapStart en la versión no publicada de una función (\$LATEST).

## Regiones admitidas

SnapStart está disponible en las siguientes Regiones de AWS:

- Este de EE. UU. (Norte de Virginia)
- Este de EE. UU. (Ohio)
- Oeste de EE. UU. (Norte de California)
- Oeste de EE. UU. (Oregón)
- África (Ciudad del Cabo)
- Asia-Pacífico (Hong Kong)
- Asia-Pacífico (Mumbai)
- Asia-Pacífico (Hyderabad)
- Asia-Pacífico (Tokio)
- Asia-Pacífico (Seúl)
- Asia-Pacífico (Osaka)

- Asia-Pacífico (Singapur)
- Asia-Pacífico (Sídney)
- Asia-Pacífico (Yakarta)
- Asia-Pacífico (Melbourne)
- Canadá (centro)
- Europa (Estocolmo)
- Europa (Fráncfort)
- Europa (Zúrich)
- Europa (Irlanda)
- Europa (Londres)
- Europa (París)
- Europa (Milán)
- Europa (España)
- Medio Oriente (EAU)
- Medio Oriente (Baréin)
- América del Sur (São Paulo)

## Consideraciones sobre compatibilidad

Con SnapStart, Lambda utiliza una única instantánea como estado inicial para varios entornos de ejecución. Si la función utiliza alguno de los siguientes factores durante la [fase de inicialización](#), es posible que deba realizar algunos cambios antes de usar SnapStart:

### Singularidad

Si el código de inicialización genera contenido único que se incluye en la instantánea, es posible que el contenido no lo sea cuando se reutilice en los entornos de ejecución. Para mantener la exclusividad al utilizar SnapStart, debe generar contenido único después de la inicialización. Esto incluye ID y secretos únicos y entropía que se utiliza para generar pseudoaleatoriedad. Para aprender a restaurar la singularidad, consulte [Control de la exclusividad con Lambda SnapStart](#).

### Conexiones de red

El estado de las conexiones que establece la función durante la fase de inicialización no está garantizado cuando Lambda vuelve a activar la función a partir de una instantánea. Valide

el estado de sus conexiones de red y vuelva a establecerlas según sean necesarias. En la mayoría de los casos, las conexiones de red que establece un SDK de AWS se reanudan automáticamente. Para otras conexiones, consulte las [prácticas recomendadas](#).

## Datos temporales

Algunas funciones descargan o inicializan datos efímeros, como credenciales temporales o marcas temporales almacenadas en caché, durante la fase de inicialización. Actualice los datos efímeros en el controlador de funciones antes de usarlos, incluso cuando no utilice SnapStart.

## Precios de SnapStart

No se aplica ningún costo adicional por usar SnapStart. Se le cobrará en función del número de solicitudes de tus funciones, del tiempo que el código tarda en ejecutarse y de la memoria configurada para la función. La duración se calcula desde el momento en que el código comienza a ejecutarse hasta que devuelve resultados o finaliza de algún otro modo, y se redondea al milisegundo más cercano.


Los cargos por la duración se aplican al código que se ejecuta en el [controlador](#) de funciones, al código de inicialización que se declara fuera del controlador, al periodo que tarda en cargarse el tiempo de ejecución (JVM) y a cualquier código que se ejecute en un [enlace de tiempo de ejecución](#). Para obtener más información sobre cómo calcula Lambda la duración, consulte [Monitoreo para Lambda SnapStart](#).

Para las funciones configuradas con SnapStart, Lambda recicla periódicamente los entornos de ejecución y vuelve a ejecutar el código de inicialización. Para mejorar la resiliencia, Lambda crea instantáneas en varias zonas de disponibilidad. Se cobran cargos cada vez que Lambda vuelve a ejecutar el código de inicialización en otra zona de disponibilidad. Para obtener más información sobre cómo Lambda calcula los cargos, consulte [Precios de AWS Lambda](#).

## Comparación entre Lambda SnapStart y la simultaneidad aprovisionada

Tanto Lambda SnapStart como la [simultaneidad aprovisionada](#) pueden reducir los inicios en frío y las latencias atípicas cuando una función escala verticalmente. SnapStart lo ayuda a mejorar el rendimiento de inicio hasta 10 veces sin costo adicional. La simultaneidad aprovisionada mantiene las funciones inicializadas y listas para responder en cuestión de milisegundos de dos dígitos. La configuración de la simultaneidad aprovisionada genera cargos para su Cuenta de AWS. Utilice la simultaneidad aprovisionada si su aplicación tiene requisitos estrictos de latencia para los inicios en frío. No puede utilizar SnapStart y la simultaneidad aprovisionada en la misma versión de la función.



 Note

SnapStart funciona mejor cuando se usa con invocaciones de funciones a escala. Es posible que las funciones que se invocan con poca frecuencia no experimenten las mismas mejoras de rendimiento.

## Recursos adicionales de

Además de leer los demás temas de este capítulo, también le recomendamos que pruebe el taller [Inicio más rápido con AWS Lambda SnapStart](#) y que vea la sesión [Inicios en frío rápidos para sus funciones de Java](#) de AWS re:Invent 2022.

## Activación y administración de Lambda SnapStart

Para utilizarla SnapStart, SnapStart actívela en una función Lambda nueva o existente. A continuación, publique e invoque una versión de la función.

### Temas

- [Activación SnapStart \(consola\)](#)
- [Activando SnapStart \(AWS CLI\)](#)
- [Activación SnapStart \(API\)](#)
- [Lambda SnapStart y estados de función](#)
- [Actualización de una instantánea](#)
- [Utilizándola con SnapStart AWS SDK for Java](#)
- [Si se usa SnapStart con AWS CloudFormation, AWS SAM, y AWS CDK](#)
- [Eliminación de instantáneas](#)

### Activación SnapStart (consola)

#### SnapStart Para activar una función

1. Abra la página de [Functions](#) (Funciones) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija Configuration (Configuración) y, a continuación, General configuration (Configuración general).
4. En el panel de General configuration (Configuración general), elija Edit (Editar).
5. En la página Editar la configuración básica, para SnapStart, seleccione Versiones publicadas.
6. Seleccione Save (Guardar).
7. [Publique una versión de la función](#). Lambda inicializa el código, crea una instantánea del entorno de ejecución inicializado y, a continuación, almacena la instantánea en caché para acceder a ella con baja latencia.
8. [Invoque la versión de la función](#).

## Activando SnapStart (AWS CLI)

SnapStart Para activar una función existente

1. Actualice la configuración de la función ejecutando el [update-function-configuration](#) comando con la `--snap-start` opción.

```
aws lambda update-function-configuration \  
  --function-name my-function \  
  --snap-start ApplyOn=PublishedVersions
```

2. Utilice el comando [publish-version](#) para publicar una versión de la función.

```
aws lambda publish-version \  
  --function-name my-function
```

3. Confirme que SnapStart está activado para la versión de la función ejecutando el [get-function-configuration](#) comando y especificando el número de versión. El siguiente ejemplo especifica la versión 1.

```
aws lambda get-function-configuration \  
  --function-name my-function:1
```

Si la respuesta muestra que [OptimizationStatuses](#) On y [State](#) isActive, SnapStart se activa y hay disponible una instantánea para la versión de función especificada.

```
"SnapStart": {  
  "ApplyOn": "PublishedVersions",  
  "OptimizationStatus": "On"  
},  
"State": "Active",
```

4. Para invocar la versión de la función, ejecute el comando [invoke](#) y especifique la versión. El siguiente ejemplo invoca la versión 1.

```
aws lambda invoke \  
  --cli-binary-format raw-in-base64-out \  
  --function-name my-function:1 \  
  --payload '{ "name": "Bob" }' \  
  response.json
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Para activarlo SnapStart al crear una función nueva

1. Ejecute el comando [create-function](#) para crear una función con la opción `--snap-start`. Para `--role`, especifique el nombre de recurso de Amazon (ARN) del [rol de ejecución](#).

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime "java21" \  
  --zip-file fileb://my-function.zip \  
  --handler my-function.handler \  
  --role arn:aws:iam::111122223333:role/lambda-ex \  
  --snap-start ApplyOn=PublishedVersions
```

2. Cree una versión con el comando [publish-version](#).

```
aws lambda publish-version \  
  --function-name my-function
```

3. Confirme que SnapStart está activada para la versión de la función ejecutando el [get-function-configuration](#) comando y especificando el número de versión. El siguiente ejemplo especifica la versión 1.

```
aws lambda get-function-configuration \  
  --function-name my-function:1
```

Si la respuesta muestra que `OptimizationStatuses` `On` y `State` `isActive`, SnapStart se activa y hay disponible una instantánea para la versión de función especificada.

```
"SnapStart": {  
  "ApplyOn": "PublishedVersions",  
  "OptimizationStatus": "On"  
},
```

```
"State": "Active",
```

- Para invocar la versión de la función, ejecute el comando [invoke](#) y especifique la versión. El siguiente ejemplo invoca la versión 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

## Activación SnapStart (API)

Para activar SnapStart

- Realice una de las acciones siguientes:
  - Cree una nueva función SnapStart activada mediante la acción de la [CreateFunction](#) API con el [SnapStart](#) parámetro.
  - Actívala SnapStart para una función existente mediante la [UpdateFunctionConfiguration](#) acción con el [SnapStart](#) parámetro.
- Publique una versión de la función con la [PublishVersion](#) acción. Lambda inicializa el código, crea una instantánea del entorno de ejecución inicializado y, a continuación, almacena la instantánea en caché para acceder a ella con baja latencia.
- Confirme que SnapStart está activada para la versión de la función mediante la [GetFunctionConfiguration](#) acción. Especifique un número de versión para confirmar que SnapStart está activada para esa versión. Si la respuesta muestra que [OptimizationStatuses](#) On y [State](#) es Active, SnapStart se activa y hay disponible una instantánea para la versión de la función especificada.

```
"SnapStart": {
  "ApplyOn": "PublishedVersions",
```

```
    "OptimizationStatus": "On"  
  },  
  "State": "Active",
```

4. Invoque la versión de la función con la acción [Invoke](#).

## Lambda SnapStart y estados de función

Los siguientes estados de función se pueden producir cuando se usa SnapStart. También pueden producirse cuando Lambda recicla periódicamente el entorno de ejecución y vuelve a ejecutar el código de inicialización de una función configurada con SnapStart

- **Pending**: Lambda está inicializando el código y tomando una instantánea del entorno de ejecución inicializado. Las invocaciones u otras acciones de API que actúen en la versión de la función producirán errores.
- **Active**: la creación de instantáneas está completa y puede invocar la función. Para usarlo SnapStart, debe invocar la versión de la función publicada, no la versión no publicada (\$LATEST).
- **Inactive**: la versión de la función no se ha invocado durante 14 días. Cuando la versión de la función se vuelve **Inactive**, Lambda elimina la instantánea. Si invoca la versión de la función transcurridos 14 días, Lambda devuelve una respuesta `SnapStartNotReadyException` y comienza a inicializar una nueva instantánea. Espere hasta que la versión de la función alcance el estado **Active** y, a continuación, vuelva a invocarla.
- **Failed**: Lambda encontró un error al ejecutar el código de inicialización o al crear la instantánea.

## Actualización de una instantánea

Lambda crea una instantánea para cada versión de función publicada. Para actualizar una instantánea, publique una nueva versión de la función. Lambda actualiza automáticamente las instantáneas con los parches de seguridad y de tiempo de ejecución más recientes.

## Utilizándola con SnapStart AWS SDK for Java

Para realizar llamadas AWS SDK desde su función, Lambda genera un conjunto efímero de credenciales al asumir el rol de ejecución de la función. Estas credenciales están disponibles como variables de entorno durante la invocación de la función. No es necesario proporcionar las credenciales de SDK directamente en el código. De forma predeterminada, la cadena de proveedores de credenciales comprueba secuencialmente cada lugar en el que puede

configurar las credenciales y elige el primero disponible, normalmente las variables de entorno (AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY, y AWS\_SESSION\_TOKEN).

#### Note

Cuando SnapStart está activado, el motor de ejecución de Java utiliza automáticamente las credenciales del contenedor (AWS\_CONTAINER\_CREDENTIALS\_FULL\_URI y AWS\_CONTAINER\_AUTHORIZATION\_TOKEN) en lugar de las variables de entorno de la clave de acceso. Esto evita que las credenciales caduquen antes de que se restablezca la función.

Si se usa SnapStart con AWS CloudFormation, AWS SAM, y AWS CDK

- AWS CloudFormation: Declare la [SnapStart](#) entidad en su plantilla.
- AWS Serverless Application Model (AWS SAM): Declare la [SnapStart](#) propiedad en su plantilla.
- AWS Cloud Development Kit (AWS CDK): Usa el [SnapStartProperty](#) tipo.

## Eliminación de instantáneas

Lambda elimina las instantáneas cuando sucede lo siguiente:

- Elimina la función o la versión de la función.
- No invoca la versión de la función durante 14 días. Tras 14 días sin una invocación, la versión de la función pasa al estado [Inactive](#) (Inactivo). Si invoca la versión de la función transcurridos 14 días, Lambda devuelve una respuesta `SnapStartNotReadyException` y comienza a inicializar una nueva instantánea. Espere hasta que la versión de la función alcance el estado [Active](#) (Activo) y, a continuación, vuelva a invocarla.

Lambda elimina todos los recursos asociados a las instantáneas eliminadas de acuerdo con el Reglamento General de Protección de Datos (RGPD).

## Control de la exclusividad con Lambda SnapStart

Cuando las invocaciones escalar verticalmente en una función de SnapStart, Lambda utiliza una instantánea exclusiva inicializada para reanudar varios entornos de ejecución. Si el código de inicialización genera contenido único que se incluye en la instantánea, es posible que el contenido no lo sea cuando se reutilice en los entornos de ejecución. Para mantener la exclusividad al utilizar SnapStart, debe generar contenido único después de la inicialización. Esto incluye ID y secretos únicos y entropía que se utiliza para generar pseudoaleatoriedad.

Sugerimos las siguientes prácticas recomendadas para ayudarlo a mantener la exclusividad en su código: Lambda también proporciona una [herramienta de escaneo SnapStart](#) de código abierto para ayudar a comprobar si el código asume que es exclusivo. Si genera datos exclusivos durante la fase de inicialización, puede utilizar un [enlace de tiempo de ejecución](#) para restaurar la exclusividad. Con los enlaces de tiempo de ejecución, puede ejecutar un código específico inmediatamente antes de que Lambda tome una instantánea o inmediatamente después de que Lambda reanude una función a partir de una instantánea.

Evite guardar el estado que depende de la exclusividad durante la inicialización.

Durante la [fase de inicialización](#) de la función, evite almacenar en caché datos destinados a ser exclusivos, por ejemplo, generar un ID exclusivo para el registro. En su lugar, le recomendamos que genere datos exclusivos dentro del controlador de funciones o que utilice un [enlace de tiempo de ejecución](#).

Example — Generar un ID exclusivo en el controlador de funciones

En el ejemplo siguiente se muestra cómo generar un UUID en el controlador de funciones.

```
import java.util.UUID;
public class Handler implements RequestHandler<String, String> {
    private static UUID uniqueSandboxId = null;
    @Override
    public String handleRequest(String event, Context context) {
        if (uniqueSandboxId == null)
            uniqueSandboxId = UUID.randomUUID();
        System.out.println("Unique Sandbox Id: " + uniqueSandboxId);
        return "Hello, World!";
    }
}
```



## Utilice generadores de números pseudoaleatorios criptográficamente seguros (CSPRNG)

Si su aplicación depende de la aleatoriedad, le recomendamos que utilice generadores de números pseudoaleatorios criptográficamente seguros. El tiempo de ejecución gestionado por Lambda para Java incluye dos CSPRNG integrados (OpenSSL 1.0.2 y `java.security.SecureRandom`) que mantienen automáticamente la aleatoriedad con SnapStart. Software que siempre obtiene números aleatorios de `/dev/random` o `/dev/urandom` también mantiene la aleatoriedad con SnapStart.

Las bibliotecas de criptografía de AWS mantienen automáticamente la aleatoriedad con SnapStart a partir de las versiones mínimas especificadas en la siguiente tabla. Si utiliza estas bibliotecas con las funciones de Lambda, asegúrese de usar las siguientes versiones mínimas o posteriores:

Library	Versión mínima compatible (x86)	Versión mínima compatible (ARM)
AWS libcrypto (AWS-LC)	1.16.0	1.30.0
AWS libcrypto FIPS	2.0.13	2.0.13

Si empaqueta las bibliotecas criptográficas anteriores con sus funciones de Lambda como dependencias transitivas a través de las siguientes bibliotecas, asegúrese de utilizar las siguientes versiones mínimas o posteriores:

Library	Versión mínima compatible (x86)	Versión mínima compatible (ARM)
AWS SDK for Java 2.x	2.23.20	2.26.12
AWS Common Runtime for Java	0.29.8	0.29.25
Amazon Corretto Crypto Provider	2.4.1	2.4.1
Amazon Corretto Crypto Provider, FIPS	2.4.1	2.4.1

## Example — java.security.Secure Random

En el siguiente ejemplo se utiliza `java.security.SecureRandom`, que genera secuencias numéricas exclusivas incluso cuando la función se restaura a partir de una instantánea.

```
import java.security.SecureRandom;
public class Handler implements RequestHandler<String, String> {
    private static SecureRandom rng = new SecureRandom();
    @Override
    public String handleRequest(String event, Context context) {
        for (int i = 0; i < 10; i++) {
            System.out.println(rng.next());
        }
        return "Hello, World!";
    }
}
```

## Herramienta de análisis de SnapStart

Lambda proporciona una herramienta de escaneo para ayudar a comprobar si el código asume que es exclusivo. La herramienta de escaneo SnapStart es un complemento de [SpotBugs](#) de código abierto que ejecuta un análisis estático en función de un conjunto de reglas. La herramienta de escaneo ayuda a identificar posibles implementaciones de código que podrían romper las suposiciones con respecto a la exclusividad. Para obtener instrucciones de instalación y una lista de las comprobaciones que realiza la herramienta de escaneo, consulte el repositorio [aws-lambda-snapstart-java-rules](#) en GitHub.

Para obtener más información sobre cómo gestionar la exclusividad con SnapStart, consulte [Inicio más rápido con AWS Lambda SnapStart](#) en el blog de computación de AWS.

## Implemente el código antes o después de las instantáneas de la función de Lambda

Puede utilizar enlaces de tiempo de ejecución para implementar el código antes de que Lambda cree una instantánea o después de que Lambda restaure una función desde una instantánea. Los enlaces de tiempo de ejecución están disponibles como parte del proyecto Coordinated Restore at Checkpoint (CRaC) de código abierto. El CRaC está en desarrollo para el [Open Java Development Kit \(OpenJDK\)](#). Para ver un ejemplo de cómo utilizar CRaC con una aplicación de referencia, visite el repositorio de [CRaC](#) en GitHub. CRaC utiliza tres elementos principales:

- **Resource**: una interfaz con dos métodos, `beforeCheckpoint()` y `afterRestore()`. Utilice estos métodos para implementar el código que desea ejecutar antes de una instantánea y después de una restauración.
- **Context** `<R extends Resource>`: para recibir notificaciones de puntos de comprobación y restauraciones, debe haber un `Resource` con un `Context`.
- **Core**: el servicio de coordinación, que proporciona el `Context` global predeterminado mediante el método estático `Core.getGlobalContext()`.

Para obtener más información sobre `Context` y `Resource`, consulte [Package org.crac](#) en la documentación de CRaC.

Siga los siguientes pasos para implementar enlaces de tiempo de ejecución con [Package org.crac](#). El tiempo de ejecución de Lambda contiene una implementación contextual de CRaC personalizada que invoca los enlaces de tiempo de ejecución antes de realizar comprobaciones y después de restaurarlos.

### Paso 1: Actualizar la configuración de compilación

Agregue la dependencia `org.crac` a la configuración de compilación. El siguiente ejemplo utiliza Gradle. Para ver ejemplos de otros sistemas de compilación, consulte la [documentación de Apache Maven](#).

```
dependencies {
    compile group: 'com.amazonaws', name: 'aws-lambda-java-core', version: '1.2.1'
    # All other project dependencies go here:
    # ...
    # Then, add the org.crac dependency:
    implementation group: 'org.crac', name: 'crac', version: '1.4.0'
```

```
}
```

## Paso 2: Actualizar del controlador de Lambda

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

Para obtener más información, consulte [Definir el controlador de las funciones de Lambda en Java](#).

El siguiente controlador de ejemplo muestra cómo ejecutar el código antes de comprobar (`beforeCheckpoint()`) y después de restaurarlo (`afterRestore()`). Este controlador también registra el Resource al Context global administrado en tiempo de ejecución.

### Note

Cuando Lambda crea una instantánea, el código de inicialización puede ejecutarse durante un máximo de 15 minutos. El límite de tiempo es de 130 segundos o el [tiempo de espera de la función configurado](#) (máximo de 900 segundos), lo que sea mayor. Los enlaces en tiempo de ejecución `beforeCheckpoint()` cuentan para el límite de tiempo del código de inicialización. Cuando Lambda restaura una instantánea, el tiempo de ejecución (JVM) debe cargarse y los enlaces de tiempo de ejecución `afterRestore()` deben completarse antes de que transcurra el tiempo de espera (10 segundos). De lo contrario, obtendrá una excepción `SnapStartTimeoutException`.

```
...
import org.crac.Resource;
import org.crac.Core;
...
public class CRaCDemo implements RequestStreamHandler, Resource {
    public CRaCDemo() {
        Core.getGlobalContext().register(this);
    }
    public String handleRequest(String name, Context context) throws IOException {
        System.out.println("Handler execution");
        return "Hello " + name;
    }
    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context)
```

```

        throws Exception {
        System.out.println("Before checkpoint");
    }
    @Override
    public void afterRestore(org.crac.Context<? extends Resource> context)
        throws Exception {
        System.out.println("After restore");
    }

```

Context mantiene solo una [WeakReference](#) para el objeto registrado. Si un [Resource](#) es una recopilación de elementos no utilizados, los enlaces de tiempo de ejecución no se ejecutan. Su código debe mantener una referencia segura para el Resource para garantizar que se ejecute el enlace de tiempo de ejecución.

Estos son dos ejemplos de patrones que se deben evitar:

Example — Objeto sin una referencia sólida

```
Core.getGlobalContext().register( new MyResource() );
```

Example — Objetos de clases anónimas

```

Core.getGlobalContext().register( new Resource() {

    @Override
    public void afterRestore(Context<? extends Resource> context) throws Exception {
        // ...
    }

    @Override
    public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
        // ...
    }

} );

```

En su lugar, mantenga una referencia sólida. En el siguiente ejemplo, el recurso registrado no es una recopilación de elementos no utilizados y los enlaces de tiempo de ejecución se ejecutan de manera uniforme.

## Example — Objeto con una referencia sólida

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent  
the registered resource from being garbage collected  
Core.getGlobalContext().register( myResource );
```

## Monitoreo para Lambda SnapStart

Puede monitorear las funciones de Lambda SnapStart con Amazon CloudWatch, AWS X-Ray y la [Acceso a datos de telemetría en tiempo real para extensiones mediante la API de telemetría](#).

### Note

Las [variables de entorno](#) `AWS_LAMBDA_LOG_GROUP_NAME` y `AWS_LAMBDA_LOG_STREAM_NAME` no están disponibles en las funciones de Lambda SnapStart.

## CloudWatch para SnapStart

Hay algunas diferencias con el formato de [flujo de registro de CloudWatch](#) para las funciones de SnapStart:

- Registros de inicialización: cuando se crea un nuevo entorno de ejecución, el REPORT no incluye el campo `Init Duration`. Esto se debe a que Lambda inicializa las funciones de SnapStart cuando se crea una versión, en lugar de durante la invocación de la función. Para las funciones de SnapStart, el campo `Init Duration` está en el registro `INIT_REPORT`. Este registro muestra los detalles de [Fase "init"](#), lo que incluye la duración de cualquier [enlace de tiempo de ejecución](#) de `beforeCheckpoint`.
- Registros de invocación: cuando se crea un nuevo entorno de ejecución, el REPORT incluye los campos `Restore Duration` y `Billed Restore Duration`:
  - `Restore Duration`: el tiempo que tarda Lambda en restaurar una instantánea, cargar el tiempo de ejecución (JVM) y ejecutar cualquier enlace `afterRestore`. El proceso de restauración de instantáneas puede incluir el tiempo dedicado a actividades fuera de la micro VM. Este tiempo se informa en `Restore Duration`.
  - `Billed Restore Duration`: el tiempo que tarda Lambda en cargar el tiempo de ejecución (JVM) y ejecutar cualquier enlace `afterRestore`. No se le cobrará por el tiempo que tarde en restaurar una instantánea.

### Note

Los cargos por la duración se aplican al código que se ejecuta en el [controlador](#) de funciones, al código de inicialización que se declara fuera del controlador, al periodo que

tarda en cargarse el tiempo de ejecución (JVM) y a cualquier código que se ejecute en un [enlace de tiempo de ejecución](#). Para obtener más información, consulte [Precios de SnapStart](#).

La duración del arranque en frío es la suma de `Restore Duration + Duration`.

El siguiente ejemplo es una consulta de Lambda Insights que devuelve los percentiles de latencia de las funciones de SnapStart. Para obtener más información sobre las consultas de Lambda Insights, consulte [Ejemplo de flujo de trabajo mediante consultas para solucionar problemas de una función](#).

```
filter @type = "REPORT"
  | parse @log /\d+:\aws\lambda\(?<function>.*)/
  | parse @message /Restore Duration: (?<restoreDuration>.*?) ms/
  | stats
count(*) as invocations,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 50) as p50,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 90) as p90,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99) as p99,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99.9) as p99.9
group by function, (ispresent(@initDuration) or ispresent(restoreDuration)) as
coldstart
  | sort by coldstart desc
```

## Seguimiento activo de X-Ray para SnapStart

Puede usar [X-Ray](#) para rastrear las solicitudes a las funciones de Lambda SnapStart. Hay algunas diferencias con los subsegmentos de X-Ray de las funciones SnapStart:

- No hay ningún subsegmento de `Initialization` para las funciones SnapStart.
- El subsegmento `Restore` muestra el tiempo que tarda Lambda en restaurar una instantánea, cargar el tiempo de ejecución (JVM) y ejecutar cualquier [enlace de tiempo de ejecución](#) de `afterRestore`. El proceso de restauración de instantáneas puede incluir el tiempo dedicado a actividades fuera de la micro VM. Esta vez se informa en el subsegmento `Restore`. No se le cobrará por el tiempo que pase fuera de la micro VM para restaurar una instantánea.

## Eventos de la API de telemetría para SnapStart

Lambda envía los siguientes eventos de SnapStart a [API de telemetría](#):



- [platform.restoreStart](#): muestra la hora en que comenzó la [fase Restore](#).
- [platform.restoreRuntimeDone](#): muestra si la fase Restore se ha realizado correctamente. Lambda envía este mensaje cuando el tiempo de ejecución envía una solicitud de API `restore/next` de tiempo de ejecución. Hay tres estados posibles: éxito, error y tiempo de espera agotado.
- [platform.restoreReport](#): muestra cuánto duró la fase Restore y cuántos milisegundos se le facturaron durante esta fase.

## Métricas de URL de función y Amazon API Gateway

Si crea una API web [con API Gateway](#), puede utilizar la métrica de [IntegrationLatency](#) para medir la latencia completa (el tiempo transcurrido entre el momento en que API Gateway transmite una solicitud al backend y el momento en que recibe una respuesta del backend).

Si utiliza la [URL de función de Lambda](#), puede utilizar la métrica [URLRequestLatency](#) para medir la latencia completa (el tiempo transcurrido entre el momento en que la URL de función recibe una solicitud y el momento en que la URL de función devuelve una respuesta).

## Modelo de seguridad para Lambda SnapStart

Lambda SnapStart admite el cifrado en reposo. Lambda cifra las instantáneas con una AWS KMS key. De forma predeterminada, Lambda utiliza una Clave administrada de AWS. Si este comportamiento predeterminado se ajusta a su flujo de trabajo, no tiene que configurar nada más. De lo contrario, puede utilizar la `--kms-key-arn` opción de la [función o el `update-function-configuration` comando `create-function`](#) para proporcionar una clave gestionada por el AWS KMS cliente. Puede hacerlo para controlar la rotación de la clave de KMS o para cumplir con los requisitos de su organización para administrar claves de KMS. Las claves administradas por el cliente ocasionan cargos de AWS KMS estándar. Para más información, consulte [Precios de AWS Key Management Service](#).

Al eliminar una SnapStart función o versión de una función, todas las Invoke solicitudes a esa función o versión de la función fallan. Lambda elimina automáticamente las instantáneas que no se invocan durante 14 días. Lambda elimina todos los recursos asociados a las instantáneas eliminadas de acuerdo con el Reglamento General de Protección de Datos (RGPD).

# Maximice el rendimiento de Lambda SnapStart

## Temas

- [Ajuste del rendimiento](#)
- [Prácticas recomendadas para redes](#)

## Ajuste del rendimiento

### Note

SnapStart funciona mejor cuando se usa con invocaciones de funciones a escala. Es posible que las funciones que se invocan con poca frecuencia no experimenten las mismas mejoras de rendimiento.

Para maximizar las ventajas de SnapStart, le recomendamos que precargue las clases que contribuyen a la latencia de inicio en el código de inicialización en lugar de hacerlo en el controlador de funciones. Esto elimina de la ruta de invocación la latencia asociada a la carga pesada de clases, lo que optimiza el rendimiento de inicio con SnapStart.

Si no puede precargar las clases durante la inicialización, le recomendamos que las precargue con invocaciones ficticias. Para ello, actualice el código del controlador de funciones, como se muestra en el siguiente ejemplo de la [función tienda de mascotas](#) del repositorio de Labs GitHub de AWS.

```
private static SpringLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;
static {
    try {
        handler =
SpringLambdaContainerHandler.getAwsProxyHandler(PetStoreSpringAppConfig.class);

        // Use the onStartup method of the handler to register the custom filter
        handler.onStartup(servletContext -> {
            FilterRegistration.Dynamic registration =
servletContext.addFilter("CognitoIdentityFilter", CognitoIdentityFilter.class);
            registration.addMappingForUrlPatterns(EnumSet.of(DispatcherType.REQUEST),
false, "/*");
        });
    }
}
```

```
// Send a fake Amazon API Gateway request to the handler to load classes
ahead of time
ApiGatewayRequestIdentity identity = new ApiGatewayRequestIdentity();
identity.setApiKey("foo");
identity.setAccountId("foo");
identity.setAccessKey("foo");

AwsProxyRequestContext reqCtx = new AwsProxyRequestContext();
reqCtx.setPath("/pets");
reqCtx.setStage("default");
reqCtx.setAuthorizer(null);
reqCtx.setIdentity(identity);

AwsProxyRequest req = new AwsProxyRequest();
req.setHttpMethod("GET");
req.setPath("/pets");
req.setBody("");
req.setRequestContext(reqCtx);

Context ctx = new TestContext();
handler.proxy(req, ctx);

} catch (ContainerInitializationException e) {
    // if we fail here. We re-throw the exception to force another cold start
    e.printStackTrace();
    throw new RuntimeException("Could not initialize Spring framework", e);
}
}
```

## Prácticas recomendadas para redes

El estado de las conexiones que establece la función durante la fase de inicialización no está garantizado cuando Lambda vuelve a activar la función a partir de una instantánea. En la mayoría de los casos, las conexiones de red que establece un SDK de AWS se reanudan automáticamente. Recomendamos que siga las siguientes prácticas recomendadas para otras conexiones.

Restablezca las conexiones de red.

Restablezca siempre las conexiones de red cuando la función se reanude a partir de una instantánea. Se recomienda restablecer las conexiones de red en el controlador de funciones. Como alternativa, puede utilizar un [enlace de tiempo de ejecución](#) de `afterRestore`.

No utilice el nombre de host como identificador único del entorno de ejecución.

Se recomienda no utilizar un `hostname` para identificar el entorno de ejecución como un nodo o contenedor único en las aplicaciones. Con SnapStart, se utiliza una sola instantánea como estado inicial para varios entornos de ejecución y todos los entornos de ejecución devuelven el mismo valor de `hostname` para `InetAddress.getLocalHost()`. Para las aplicaciones que requieren una identidad de entorno de ejecución o un valor de `hostname` único, se recomienda generar un ID único en el controlador de funciones. O bien, utilice un [enlace de tiempo de ejecución](#) de `afterRestore` para generar un ID único y, a continuación, utilice el ID único como identificador del entorno de ejecución.

Evite vincular conexiones a puertos de origen fijos.

Se recomienda evitar vincular las conexiones de red a puertos de origen fijos. Las conexiones se restablecen cuando se reanuda una función a partir de una instantánea y las conexiones de red que están enlazadas a un puerto de origen fijo pueden fallar.

Evite utilizar la caché de DNS de Java.

Las funciones de Lambda ya almacenan en caché las respuestas de DNS. Si utiliza otra caché de DNS con SnapStart, es posible que se agoten los tiempos de espera de conexión cuando la función se reanude a partir de una instantánea.

La clase `java.util.logging.Logger` puede habilitar indirectamente la caché de DNS de la JVM. Para anular la configuración predeterminada, defina [networkaddress.cache.ttl](#) en 0 antes de inicializar `logger`. Ejemplo:

```
public class MyHandler {
    // first set TTL property
    static{
        java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
    }
    // then instantiate logger
    var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

Para evitar errores de `UnknownHostException`, se recomienda establecer `networkaddress.cache.negative.ttl` en 0. Puede establecer esta propiedad para una función de Lambda con la variable de entorno `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

Al deshabilitar la caché de DNS de la JVM, no se deshabilita la caché de DNS administrada de Lambda.

# Personalice el comportamiento de inicio del tiempo de ejecución de Java para funciones de Lambda

En esta página, se describe la configuración específica de las funciones de Java en AWS Lambda. Puede utilizar esta configuración para personalizar el comportamiento de inicio del tiempo de ejecución de Java. Esto puede reducir la latencia general de las funciones y mejorar su rendimiento general, sin tener que modificar ningún código.

## Secciones

- [Cómo entender la variable de entorno `JAVA\_TOOL\_OPTIONS`](#)

## Cómo entender la variable de entorno `JAVA_TOOL_OPTIONS`

En Java, Lambda admite la variable de entorno `JAVA_TOOL_OPTIONS` para establecer variables de línea de comandos adicionales. Puede utilizar esta variable de entorno de varias maneras, como para personalizar la configuración de compilación por niveles. En el siguiente ejemplo, se muestra cómo utilizar la variable de entorno `JAVA_TOOL_OPTIONS` en este caso de uso.

### Ejemplo: cómo personalizar la configuración de compilación por niveles

La compilación por niveles es una característica de la máquina virtual Java (JVM). Puede utilizar configuraciones específicas de compilación por niveles para aprovechar al máximo la compilación en tiempo de ejecución (JIT) de la JVM. Por lo general, el compilador C1 está optimizado para un inicio rápido. El compilador C2 está optimizado para obtener el mejor rendimiento general, pero también utiliza más memoria y tarda más en lograrlo.

Hay 5 niveles diferentes de compilación por niveles. En el nivel 0, la JVM interpreta el código de bytes de Java. En el nivel 4, la JVM usa el compilador C2 para analizar los datos de creación de perfiles recopilados durante el inicio de la aplicación. Con el tiempo, monitorea el uso del código para identificar las mejores optimizaciones.

La personalización del nivel de compilación por niveles puede ayudarlo a reducir la latencia de inicio en frío de la función de Java. Por ejemplo, defina el nivel de compilación por niveles en 1 para que la JVM utilice el compilador C1. Este compilador produce rápidamente código nativo optimizado, pero no genera ningún dato de creación de perfiles y nunca usa el compilador C2.

En el tiempo de ejecución de Java 17, el indicador de JVM para la compilación por niveles está configurado para detenerse en el nivel 1 de forma predeterminada. Para el tiempo de ejecución de

Java 11 y versiones anteriores, puede establecer el nivel de compilación por niveles en 1 mediante los siguientes pasos:

Para personalizar la configuración de compilación por niveles (consola)

1. Abra la página [Funciones](#) en la consola de Lambda.
2. Elija la función de Java para la que desea personalizar la compilación por niveles.
3. Elija la pestaña Configuración y, a continuación, elija Variables de entorno en el menú de la izquierda.
4. Elija Editar.
5. Elija Add environment variable (Añadir variable de entorno).
6. Para la clave, ingrese JAVA\_TOOL\_OPTIONS. Para el valor, ingrese `-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.

## Edit environment variables

**Environment variables**

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
JAVA_TOOL_OPTIONS	-XX:+TieredCompilation -XX:TieredStopAtLevel=1	Remove

[Add environment variable](#)

► Encryption configuration

Cancel **Save**

7. Seleccione Guardar.



**Note**

También puede utilizar Lambda SnapStart para mitigar los problemas de arranque en frío. SnapStart utiliza instantáneas almacenadas en caché de su entorno de ejecución para mejorar de forma significativa el rendimiento de inicio. Para obtener más información sobre las características, las limitaciones y las regiones admitidas de SnapStart, consulte [Mejora del rendimiento de inicio con Lambda SnapStart](#).

## Ejemplo: personalización del comportamiento de GC mediante JAVA\_TOOL\_OPTIONS

Los tiempos de ejecución de Java 11 utilizan el recopilador de elementos no utilizados (GC) [en serie](#) para la recopilación de elementos no utilizados. De forma predeterminada, los tiempos de ejecución de Java 17 también utilizan el GC en serie. Sin embargo, con Java 17 también puede usar la variable de entorno JAVA\_TOOL\_OPTIONS para cambiar el GC predeterminado. Puede elegir entre el GC en paralelo y el [GC Shenandoah](#).

Por ejemplo, si su carga de trabajo utiliza más memoria y varias CPU, considere la posibilidad de utilizar el GC en paralelo para obtener un mejor rendimiento. Para ello, agregue lo siguiente al valor de la variable de entorno JAVA\_TOOL\_OPTIONS:

```
-XX:+UseParallelGC
```

# Uso del objeto de contexto Lambda para recuperar información de funciones de Java

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades que facilitan información acerca de la invocación, la función y el entorno de ejecución.

## Métodos de context

- `getRemainingTimeInMillis()`: devuelve el número de milisegundos que quedan antes del tiempo de espera de la ejecución.
- `getFunctionName()`: el nombre de la función de Lambda.
- `getFunctionVersion()`: devuelve la [version](#) de la función.
- `getInvokedFunctionArn()`: devuelve el nombre de recurso de Amazon (ARN) que se utiliza para invocar la función. Indica si el invocador especificó un número de versión o alias.
- `getMemoryLimitInMB()`: devuelve la cantidad de memoria asignada a la función.
- `getAwsRequestId()`: devuelve el identificador de la solicitud de invocación.
- `getLogGroupName()`: devuelve el grupo de registros de para la función.
- `getLogStreamName()`: devuelve el flujo de registro de la instancia de la función.
- `getIdentity()`: (aplicaciones móviles) devuelve la información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
- `getClientContext()`: (aplicaciones móviles) devuelve el contexto de cliente proporcionado a Lambda por la aplicación cliente.
- `getLogger()`: devuelve el [objeto logger](#) para la función.

En el ejemplo siguiente, se muestra una función que utiliza el objeto contextual para acceder al registrador de Lambda.

## Example [Handler.java](#)

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
```

```
import com.amazonaws.services.lambda.runtime.RequestHandler;  
  
import java.util.Map;  
  
// Handler value: example.Handler  
public class Handler implements RequestHandler<Map<String,String>, Void>{  
  
    @Override  
    public Void handleRequest(Map<String,String> event, Context context)  
    {  
        LambdaLogger logger = context.getLogger();  
        logger.log("EVENT TYPE: " + event.getClass());  
        return null;  
    }  
}
```

La función registra el tipo de clase del evento entrante antes de volver a null.

#### Example salida del registro

```
EVENT TYPE: class java.util.LinkedHashMap
```

La interfaz del objeto contextual está disponible en la biblioteca [aws-lambda-java-core](#). Puede implementar esta interfaz para crear una clase contextual para las pruebas. En el ejemplo siguiente, se muestra una clase contextual que devuelve valores ficticios para la mayoría de las propiedades y un registrador de pruebas en funcionamiento.

#### Example [src/test/java/example/TestContext.java](#)

```
package example;  
  
import com.amazonaws.services.lambda.runtime.Context;  
import com.amazonaws.services.lambda.runtime.CognitoIdentity;  
import com.amazonaws.services.lambda.runtime.ClientContext;  
import com.amazonaws.services.lambda.runtime.LambdaLogger;  
  
public class TestContext implements Context{  
  
    public TestContext() {}  
    public String getAwsRequestId(){  
        return new String("495b12a8-xmpl-4eca-8168-160484189f99");  
    }  
}
```

```
public String getLogGroupName(){
    return new String("/aws/lambda/my-function");
}
public String getLogStreamName(){
    return new String("2020/02/26/[$LATEST]704f8dxmla04097b9134246b8438f1a");
}
public String getFunctionName(){
    return new String("my-function");
}
public String getFunctionVersion(){
    return new String("$LATEST");
}
public String getInvokedFunctionArn(){
    return new String("arn:aws:lambda:us-east-2:123456789012:function:my-function");
}
public CognitoIdentity getIdentity(){
    return null;
}
public ClientContext getClientContext(){
    return null;
}
public int getRemainingTimeInMillis(){
    return 300000;
}
public int getMemoryLimitInMB(){
    return 512;
}
public LambdaLogger getLogger(){
    return new TestLogger();
}
}
```

Para obtener más información sobre los registros, consulte [Registro y supervisión de las funciones de Lambda de Java](#).

## Contexto en aplicaciones de ejemplo

El repositorio de GitHub para esta guía contiene aplicaciones de ejemplo en las que se muestra el uso del objeto contextual. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación y la limpieza, una plantilla de AWS Serverless Application Model (AWS SAM) y recursos de soporte.

## Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

# Registro y supervisión de las funciones de Lambda de Java

AWS Lambda supervisa de forma automática funciones de Lambda y envía entradas de registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación y otros resultados del código de su función al flujo de registro. Para obtener más información acerca de Registros de CloudWatch, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Para generar registros desde el código de función, puede utilizar los métodos de [java.lang.System](#) o cualquier módulo de registro que escriba en `stdout` o `stderr`.

## Secciones

- [Crear una función que devuelve registros](#)
- [Uso de los controles de registro avanzados de Lambda con Java](#)
- [Implementación del registro avanzado con Log4j2 y SLF4J](#)
- [Uso de otras herramientas y bibliotecas de registro](#)
- [Uso de Powertools para AWS Lambda \(Java\) y AWS SAM para el registro estructurado](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)
- [Código de registro de ejemplo](#)

## Crear una función que devuelve registros

Para generar registros desde el código de función, puede utilizar los métodos de [java.lang.System](#) o cualquier módulo de registro que escriba en `stdout` o `stderr`. La biblioteca [aws-lambda-java-core](#) tiene una clase de registrador llamada `LambdaLogger` a la que se puede acceder desde el objeto contextual. La clase de registrador admite registros multilínea.

En el ejemplo siguiente se utiliza el registrador `LambdaLogger` proporcionado por el objeto contextual.

## Example Handler.java

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Object, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Object event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("SUCCESS");
        // log execution details
        logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        logger.log("CONTEXT: " + gson.toJson(context));
        // process event
        logger.log("EVENT: " + gson.toJson(event));
        return response;
    }
}
```

## Example formato de registro

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
EVENT:
{
    "records": [
        {
            "messageId": "19dd0b57-xmpl-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
```

```
    ...  
  }  
]  
}  
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0  
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed  
Duration: 200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

El tiempo de ejecución de Java registra las líneas START, END y REPORT de cada invocación. La línea del informe proporciona los siguientes detalles:

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.
- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- TraceId de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

## Uso de los controles de registro avanzados de Lambda con Java

Para tener más control sobre cómo se registran, procesan y consumen los registros de las funciones, puede configurar las siguientes opciones de registro para los tiempos de ejecución de Java admitidos:

- Formato de registro: seleccione entre texto sin formato y el formato JSON estructurado para los registros de su función
- Nivel de registro: para los registros en formato JSON, elija el nivel de detalle de los registros que Lambda envía a CloudWatch, como ERROR, DEBUG o INFO
- Grupo de registro: elija el grupo de registro de CloudWatch al que su función envía los registros



Para obtener más información sobre estas opciones de registro e instrucciones sobre cómo configurar la función para utilizarlas, consulte [the section called “Configuración de registros de funciones”](#).

Para utilizar las opciones de formato de registro y nivel de registro con las funciones de Lambda de Java, consulte las instrucciones en las siguientes secciones.

## Uso del formato de registro JSON estructurado con Java

Si selecciona JSON para el formato de registro de la función, Lambda enviará los registros de salida utilizando la clase `LambdaLogger` como JSON estructurado. Cada objeto de registro JSON contiene, por lo menos, cuatro pares clave-valor con las siguientes claves:

- `"timestamp"`: la hora en que se generó el mensaje de registro
- `"level"`: el nivel de registro asignado al mensaje
- `"message"`: el contenido del mensaje de registro
- `"AWSrequestId"`: el ID de solicitud único para la invocación de la función

Según el método de registro que utilice, las salidas de registro de la función capturadas en formato JSON también pueden contener pares de clave-valor adicionales.

Para asignar un nivel a los registros que cree con el registrador `LambdaLogger`, debe proporcionar un argumento `LogLevel` en el comando de registro, como se muestra en el siguiente ejemplo.

### Example Código de registro Java

```
LambdaLogger logger = context.getLogger();
logger.log("This is a debug log", LogLevel.DEBUG);
```

Este registro generado por este código de ejemplo sería capturado en los Registros de CloudWatch de la siguiente manera:

### Example Entrada de registro JSON

```
{
  "timestamp": "2023-11-01T00:21:51.358Z",
  "level": "DEBUG",
  "message": "This is a debug log",
  "AWSrequestId": "93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Si no asigna un nivel a la salida del registro, Lambda asignará el nivel INFO automáticamente.

Si su código ya usa otra biblioteca de registro para generar registros con JSON estructurado, no necesita hacer ningún cambio. Lambda no codifica dos veces ningún registro que ya esté codificado en JSON. Incluso si configura su función para utilizar el formato de registro JSON, las salidas del registro aparecen en CloudWatch en la estructura JSON que defina.

## Uso del filtrado a nivel de registro con Java

Para que AWS Lambda filtre los registros de las aplicaciones según su nivel de registro, la función debe usar registros con formato JSON. Puede lograr esto de dos maneras:

- Cree salidas de registro con el estándar `LambdaLogger` y configure su función para que utilice el formato de registro JSON. Lambda, a continuación, filtra los resultados del registro mediante el par clave-valor de “nivel” del objeto JSON que se describe en [the section called “Uso del formato de registro JSON estructurado con Java”](#). Para obtener información sobre cómo configurar el formato de registro de la función, consulte [the section called “Configuración de registros de funciones”](#).
- Utilice otra biblioteca o método de registro para crear registros estructurados en JSON en su código que incluyan un par clave-valor “nivel” que defina el nivel de la salida del registro. También puede utilizar cualquier biblioteca de registro que pueda escribir registros JSON en `stdout` o `stderr`. Por ejemplo, puede utilizar Powertools para AWS Lambda o el paquete Log4j2 para generar salidas de registros estructurados JSON a partir de su código. Consulte [the section called “Uso de Powertools para AWS Lambda \(Java\) y AWS SAM para el registro estructurado”](#) y [the section called “Implementación del registro avanzado con Log4j2 y SLF4J”](#) para obtener más información.

Cuando configura la función para que utilice el filtrado a nivel de registro, debe seleccionar una de las siguientes opciones para el nivel de registros que desea que Lambda envíe a los Registros de CloudWatch:

Nivel de registro	Uso estándar
TRACE (más detallado)	La información más detallada que se utiliza para rastrear la ruta de ejecución del código
DEBUG	Información detallada para la depuración del sistema

Nivel de registro	Uso estándar
INFO	Mensajes que registran el funcionamiento normal de su función
WARN	Mensajes sobre posibles errores que pueden provocar un comportamiento inesperado si no se abordan
ERROR	Mensajes sobre problemas que impiden que el código funcione según lo esperado
FATAL (menos detallado)	Mensajes sobre errores graves que hacen que la aplicación deje de funcionar

Para que Lambda filtre los registros de la función, también debe incluir un par clave-valor "timestamp" en la salida del registro JSON. La hora debe especificarse con un formato de marca de tiempo [RFC 3339](#) válido. Si no proporciona una marca de tiempo válida, Lambda asignará al registro el nivel INFO y agregará una marca de tiempo por usted.

Lambda envía los registros del nivel seleccionado y en un nivel inferior a CloudWatch. Por ejemplo, si configura un nivel de registro de WARN, Lambda enviará los registros correspondientes a los niveles WARN, ERROR y FATAL.

## Implementación del registro avanzado con Log4j2 y SLF4J

### Note

AWS Lambda no incluye Log4j2 en los tiempos de ejecución administrados ni en las imágenes de contenedores base. Por lo tanto, no se ven afectados por los problemas descritos en CVE-2021-44228, CVE-2021-45046 y CVE-2021-45105.

En los casos en los que una función de cliente incluya una versión de Log4j2 afectada, hemos aplicado cambios a los [tiempos de ejecución administrados](#) de Java y las [imágenes de contenedores base](#) de Lambda para poder mitigar los problemas de CVE-2021-44228, CVE-2021-45046 y CVE-2021-45105. Como resultado de este cambio, los clientes que utilizan Log4J2 pueden ver una entrada de registro adicional, similar a "Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)". Cualquier cadena de registro que haga

referencia al asignador `jndi` en la salida de Log4J2 se reemplazará por “Patched JndiLookup::lookup()”.

Independientemente de este cambio, recomendamos encarecidamente a todos los clientes cuyas funciones incluyan Log4j2 que actualicen a la última versión. En concreto, los clientes que utilicen la biblioteca `aws-lambda-java-log4j2` en sus funciones deben actualizar a la versión 1.5.0 (o posterior) y volver a implementarlas. Esta versión actualiza las dependencias de la utilidad Log4j2 subyacentes a la versión 2.17.0 (o posterior). El binario `aws-lambda-java-log4j2` actualizado está disponible en el [repositorio de Maven](#) y su código fuente está disponible en [GitHub](#).

Por último, tenga en cuenta que las bibliotecas relacionadas con `aws-lambda-java-log4j` (v1.0.0 o 1.0.1) no deben usarse por ningún motivo. Estas bibliotecas están relacionadas con la versión 1.x de `log4j`, que dejó de estar disponible en 2015. Las bibliotecas no son compatibles, no se mantienen ni están parcheadas y tienen vulnerabilidades de seguridad conocidas.

Para personalizar la salida del registro, posibilitar el registro durante las pruebas unitarias y registrar las llamadas del AWS SDK, utilice Apache Log4j2 con SLF4J. Log4j es una biblioteca de registros para programas Java que permite configurar diversos niveles de registro y utilizar bibliotecas de `append`. SLF4J es una biblioteca de fachadas que permite cambiar la biblioteca empleada sin modificar el código de la función.

Para agregar el ID de la solicitud a los registros de la función, use el `append` de la biblioteca [aws-lambda-java-log4j2](#).

Example [src/main/resources/log4j2.xml](#): configuración de `append`.

```
<Configuration>
  <Appenders>
    <Lambda name="Lambda" format="{env:AWS_LAMBDA_LOG_FORMAT:-TEXT}">
      <LambdaTextFormat>
        <PatternLayout>
          <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n </
pattern>
        </PatternLayout>
      </LambdaTextFormat>
      <LambdaJSONFormat>
        <JsonTemplateLayout eventTemplateUri="classpath:LambdaLayout.json" />
      </LambdaJSONFormat>
    </Lambda>
  </Appenders>
</Configuration>
```

```

</Appenders>
<Loggers>
  <Root level="${env:AWS_LAMBDA_LOG_LEVEL:-INFO}">
    <AppenderRef ref="Lambda"/>
  </Root>
  <Logger name="software.amazon.awssdk" level="WARN" />
  <Logger name="software.amazon.awssdk.request" level="DEBUG" />
</Loggers>
</Configuration>

```

Puede decidir cómo se configuran sus registros de Log4j2 para las salidas de texto plano o JSON especificando un diseño debajo en las etiquetas `<LambdaTextFormat>` y `<LambdaJSONFormat>`.

Con esta configuración, en el modo texto, en cada línea se antepone la fecha, la hora, el ID de la solicitud, el nivel de registro y el nombre de la clase. En el modo JSON, `<JsonTemplateLayout>` se usa con una configuración que se incluye junto con la biblioteca `aws-lambda-java-log4j2`.

SLF4J es una biblioteca de fachadas para los registros de código Java. En el código de la función, se utiliza la fábrica del registrador de SLF4J para recuperar un registrador con métodos para niveles de registro como `info()` y `warn()`. En la configuración de compilación, puede incluir la biblioteca de registros y el adaptador de SLF4J en el classpath. Si cambia las bibliotecas en la configuración de compilación, puede cambiar el tipo de registrador sin necesidad de modificar el código de la función. Es necesario utilizar SLF4J para capturar registros de SDK para Java.

En el siguiente código de ejemplo, la clase de controlador utiliza SLF4J para recuperar un registrador.

Example [src/main/java/example/HandlerS3.java](#): registro con SLF4J

```

package example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;

import static org.apache.logging.log4j.CloseableThreadContext.put;

public class HandlerS3 implements RequestHandler<S3Event, String>{

```

```

private static final Logger logger = LoggerFactory.getLogger(HandlerS3.class);

@Override
public String handleRequest(S3Event event, Context context) {
    for(var record : event.getRecords()) {
        try (var loggingCtx = put("awsRegion", record.getAwsRegion())) {
            loggingCtx.put("eventName", record.getEventName());
            loggingCtx.put("bucket", record.getS3().getBucket().getName());
            loggingCtx.put("key", record.getS3().getObject().getKey());

            logger.info("Handling s3 event");
        }
    }

    return "Ok";
}
}

```

Este código produce salidas del registro similares a las siguientes:

Example formato de registro

```

{
  "timestamp": "2023-11-15T16:56:00.815Z",
  "level": "INFO",
  "message": "Handling s3 event",
  "logger": "example.HandlerS3",
  "AWSRequestId": "0bced576-3936-4e5a-9dcd-db9477b77f97",
  "awsRegion": "eu-south-1",
  "bucket": "java-logging-test-input-bucket",
  "eventName": "ObjectCreated:Put",
  "key": "test-folder/"
}

```

La configuración de compilación toma dependencias del tiempo de ejecución del appender de Lambda y del adaptador de SLF4J, y dependencias de implementación de Log4J2.

Example build.gradle: dependencias de registro

```

dependencies {
    ...
    'com.amazonaws:aws-lambda-java-log4j2:[1.6.0,)',
    'com.amazonaws:aws-lambda-java-events:[3.11.3,)',
}

```

```
'org.apache.logging.log4j:log4j-layout-template-json:[2.17.1,)',  
'org.apache.logging.log4j:log4j-slf4j2-impl:[2.19.0,)',  
...  
}
```

Cuando ejecuta el código localmente para realizar pruebas, el objeto contextual con el registrador de Lambda no está disponible y no hay ningún ID de solicitud que el `appender` de Lambda pueda utilizar. Para ver ejemplos de configuraciones de prueba, consulte las aplicaciones de la siguiente sección.

## Uso de otras herramientas y bibliotecas de registro

[Powertools para AWS Lambda \(Java\)](#) es un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores. La [utilidad de registro](#) proporciona un logger optimizado para Lambda que incluye información adicional sobre el contexto de la función en todas las funciones con un resultado estructurado como JSON. Utilice esta utilidad para hacer lo siguiente:

- Capturar campos clave del contexto de Lambda, arranque en frío y resultados de registro de estructuras como JSON
- Registrar los eventos de invocación de Lambda cuando se le indique (desactivado de forma predeterminada)
- Imprimir todos los registros solo para un porcentaje de las invocaciones mediante el muestreo de registros (desactivado de forma predeterminada)
- Agregar claves adicionales al registro estructurado en cualquier momento
- Utilizar un formateador de registros personalizado (traiga su propio formateador) para generar registros en una estructura compatible con el RFC de registro de su organización

## Uso de Powertools para AWS Lambda (Java) y AWS SAM para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Java con módulos de [Powertools para AWS Lambda \(Java\)](#)~ integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway,

la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Java 11
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla “Hola, mundo” de Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type Zip --runtime java11 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima `Enter`.

### Note

En `HelloWorldFunction` es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```



## 6. Invoque el punto de conexión de la API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

## 7. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

El resultado del registro tendrá este aspecto:

```
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.095000
  INIT_START Runtime Version: java:11.v15    Runtime Version ARN: arn:aws:lambda:eu-
central-1::runtime:0a25e3e7a1cc9ce404bc435eeb2ad358d8fa64338e618d0c224fe509403583ca
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.114000
  Picked up JAVA_TOOL_OPTIONS: -XX:+TieredCompilation -XX:TieredStopAtLevel=1
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.793000
  Transforming org/apache/logging/log4j/core/lookup/JndiLookup
(lambdainternal.CustomerClassLoader@1a6c5a9e)
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:35.252000
  START RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765 Version: $LATEST
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.531000 {
  "_aws": {
    "Timestamp": 1675416276051,
    "CloudWatchMetrics": [
      {
        "Namespace": "sam-app-powerools-java",
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      },
      "Dimensions": [
        [
          "Service",
          "FunctionName"
        ]
      ]
    }
  }
}
```

```

    ]
  }
]
},
"function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
"traceId":
"Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
"FunctionName": "sam-app-HelloWorldFunction-y9Iu1FLJJBGD",
"functionVersion": "$LATEST",
"ColdStart": 1.0,
"Service": "service_undefined",
"logStreamId": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
"executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.974000 Feb
03, 2023 9:24:36 AM com.amazonaws.xray.AWSXRayRecorder <init>
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000 Feb
03, 2023 9:24:36 AM com.amazonaws.xray.config.DaemonConfiguration <init>
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000
INFO: Environment variable AWS_XRAY_DAEMON_ADDRESS is set. Emitting to daemon on
address XXXX.XXXX.XXXX.XXXX:2000.
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.331000
09:24:37.294 [main] INFO helloworld.App - {"version":null,"resource":"/
hello","path":"/hello/","httpMethod":"GET","headers":{"Accept":["*/
*"],"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-
Viewer":["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-
SmartTV-Viewer":["false"],"CloudFront-Is-Tablet-Viewer":["false"],"CloudFront-
Viewer-ASN":["16509"],"CloudFront-Viewer-Country":["IE"],"Host":["XXXX.execute-
api.eu-central-1.amazonaws.com"],"User-Agent":["curl/7.86.0"],"Via":["2.0
f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":["t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtivV1Q=="],"X-
Amzn-Trace-Id":["Root=1-63dcd2d1-25f90b9d1c753a783547f4dd"],"X-Forwarded-
For":["XX.XXX.XXX.XX, XX.XXX.XXX.XX"],"X-Forwarded-Port":["443"],"X-
Forwarded-Proto":["https"],"multiValueHeaders":{"Accept":["*/
*"],"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-Viewer":
["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-SmartTV-
Viewer":["false"],"CloudFront-Is-Tablet-Viewer":["false"],"CloudFront-Viewer-
ASN":["16509"],"CloudFront-Viewer-Country":["IE"],"Host":["XXXX.execute-
api.eu-central-1.amazonaws.com"],"User-Agent":["curl/7.86.0"],"Via":["2.0
f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":["t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtivV1Q=="],"X-
Amzn-Trace-Id":["Root=1-63dcd2d1-25f90b9d1c753a783547f4dd"],"X-Forwarded-
For":["XXX, XXX"],"X-Forwarded-Port":["443"],"X-Forwarded-Proto":
["https"]},"queryStringParameters":null,"multiValueQueryStringParameters":null,"pathParamet

```

```

{"accountId":"XXX","stage":"Prod","resourceId":"at73a1","requestId":"ba09ecd2-
acf3-40f6-89af-fad32df67597","operationName":null,"identity":
{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"apiK
hello","httpMethod":"GET","apiId":"XXX","path":"/Prod/
hello/","authorizer":null},"body":null,"isBase64Encoded":false}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.351000
09:24:37.351 [main] INFO helloworld.App - Retrieving https://
checkip.amazonaws.com
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.313000 {
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
  "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "xray_trace_id": "1-63dcd2d1-25f90b9d1c753a783547f4dd",
  "functionVersion": "$LATEST",
  "Service": "service_undefined",
  "logStreamId": "2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000 END
RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765
2023/02/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000
REPORT RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765    Duration: 4118.98 ms
Billed Duration: 4119 ms    Memory Size: 512 MB    Max Memory Used: 152 MB    Init
Duration: 1155.47 ms
XRAY TraceId: 1-63dcd2d1-25f90b9d1c753a783547f4dd    SegmentId: 3a028fee19b895cb
Sampled: true

```

- Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Administración de retención de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros de forma indefinida, elimine el grupo de registros o configure un periodo de retención después del cual CloudWatch los eliminará de forma automática. Para configurar la retención de registros, agregue lo siguiente a la plantilla de AWS SAM:

```
Resources:
  HelloWorldFunction:
```

```
Type: AWS::Serverless::Function
Properties:
  # Omitting other properties

LogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
    RetentionInDays: 7
```

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgt0GNkYS0yOTc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad `base64` está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar `base64 -D`.

Example `get-logs.sh` script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza `sed` para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando `get-log-events`.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como `get-logs.sh`.

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

## Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
```

```

        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

## Código de registro de ejemplo

El repositorio de GitHub para esta guía contiene aplicaciones de ejemplo en las que se muestra el uso de diferentes configuraciones de registro. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación y la limpieza, una plantilla de AWS SAM y recursos de soporte.

### Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.



- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

En la aplicación de ejemplo `java-basic`, se muestra una configuración de registro mínima que admite pruebas de registro. El código del controlador utiliza el registrador de `LambdaLogger` proporcionado por el objeto contextual. En el caso de las pruebas, la aplicación utiliza una clase personalizada de `TestLogger` que implementa la interfaz `LambdaLogger` con un registrador de `Log4j2`. `SLF4J` se utiliza como fachada para garantizar la compatibilidad con el SDK de AWS. Las bibliotecas de registro se excluyen de la salida de compilación para mantener un tamaño de paquete de implementación pequeño.

# Instrumentación del código Java en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas dos bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK para Java](#): un SDK para generar y enviar datos de seguimiento a X-Ray.
- [Powertools para AWS Lambda\(Java\)](#): un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de Powertools para AWS Lambda \(Java\) y AWS SAM para el seguimiento](#)
- [Uso de Powertools para AWS Lambda \(Java\) y el AWS CDK para el seguimiento](#)
- [Uso de ADOT para instrumentar las funciones de Java](#)
- [Uso del X-Ray SDK para instrumentar las funciones de Java](#)
- [Activación del seguimiento con la consola de Lambda](#)
- [Activación del seguimiento con la API de Lambda](#)

- [Activación del seguimiento con AWS CloudFormation](#)
- [Interpretación de un seguimiento de X-Ray](#)
- [Almacenamiento de dependencias de tiempo de ejecución en una capa \(X-Ray SDK\)](#)
- [Seguimiento de X-Ray en aplicaciones de ejemplo \(X-Ray SDK\)](#)

## Uso de Powertools para AWS Lambda (Java) y AWS SAM para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Java con módulos de [Powertools para AWS Lambda \(Java\)](#) integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Java 11
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

### Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla “Hola, mundo” de Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type Zip --runtime java11 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima **Enter**.

**Note**

En `HelloWorldFunction` es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque el punto de conexión de la API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

7. Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
New XRay Service Graph  
Start time: 2023-02-03 14:31:48+01:00  
End time: 2023-02-03 14:31:48+01:00  
Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-y9Iu1FLJJBGD -  
Edges: []  
Summary_statistics:  
  - total requests: 1  
  - ok count(2XX): 1  
  - error count(4XX): 0  
  - fault count(5XX): 0
```

```

- total response time: 5.587
Reference Id: 1 - client - sam-app-HelloWorldFunction-y9Iu1FLJJBGD - Edges: [0]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 3] at (2023-02-03T14:31:48.500000) with id
(1-63dd0cc4-3c869dec72a586875da39777) and duration (5.603s)
- 5.587s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD [HTTP: 200]
- 4.053s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD
- 1.181s - Initialization
- 4.037s - Invocation
- 1.981s - ## handleRequest
  - 1.840s - ## getPageContents
- 0.000s - Overhead

```

8. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Uso de Powertools para AWS Lambda (Java) y el AWS CDK para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de Java con módulos de [Powertools para AWS Lambda \(Java\)](#) integrados mediante AWS CDK. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje hello world.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Java 11

- [Versión 2 de la AWS CLI](#)
- [Versión 2 de la AWS CDK](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyecto para la nueva aplicación.

```
mkdir hello-world
cd hello-world
```

2. Inicialice la aplicación.

```
cdk init app --language java
```

3. Ejecute el siguiente comando para crear el proyecto de Maven:

```
mkdir app
cd app
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Abra `pom.xml` en el directorio `hello-world\app\Function` y sustituya el código existente por el siguiente código, el cual incluye dependencias y complementos de Maven para Powertools.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>helloworld</groupId>
  <artifactId>Function</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Function</name>
  <url>http://maven.apache.org</url>
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
```

```
<log4j.version>2.17.2</log4j.version>
</properties>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>software.amazon.lambda</groupId>
    <artifactId>powertools-tracing</artifactId>
    <version>1.3.0</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.lambda</groupId>
    <artifactId>powertools-metrics</artifactId>
    <version>1.3.0</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.lambda</groupId>
    <artifactId>powertools-logging</artifactId>
    <version>1.3.0</version>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-core</artifactId>
    <version>1.2.2</version>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-events</artifactId>
    <version>3.11.1</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>aspectj-maven-plugin</artifactId>
      <version>1.14.0</version>
      <configuration>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```

    <complianceLevel>${maven.compiler.target}</complianceLevel>
    <aspectLibraries>
      <aspectLibrary>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-tracing</artifactId>
      </aspectLibrary>
      <aspectLibrary>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-metrics</artifactId>
      </aspectLibrary>
      <aspectLibrary>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-logging</artifactId>
      </aspectLibrary>
    </aspectLibraries>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.4.1</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCache
          </transformer>
        </transformers>
        <createDependencyReducedPom>>false</
createDependencyReducedPom>
        <finalName>function</finalName>

```



```

        </configuration>
    </execution>
</executions>
<dependencies>
    <dependency>
        <groupId>com.github.edwgiz</groupId>
        <artifactId>maven-shade-plugin.log4j2-cachefile-
transformer</artifactId>
        <version>2.15</version>
    </dependency>
</dependencies>
</plugin>
</plugins>
</build>
</project>

```

5. Cree el directorio `hello-world\app\src\main\resource` y cree `log4j.xml` para la configuración del registro.

```

mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml

```

6. Abra `log4j.xml` y agregue el siguiente código.

```

<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="JsonAppender" target="SYSTEM_OUT">
            <JsonTemplateLayout
eventTemplateUri="classpath:LambdaJsonLayout.json" />
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="JsonLogger" level="INFO" additivity="false">
            <AppenderRef ref="JsonAppender"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="JsonAppender"/>
        </Root>
    </Loggers>
</Configuration>

```

7. Abra `App.java` desde el directorio `hello-world\app\Function\src\main\java\helloworld` y sustituya el código existente por el siguiente código. Se trata del código de la función de Lambda.

```
package helloworld;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.metrics.Metrics;
import software.amazon.lambda.powertools.tracing.CaptureMode;
import software.amazon.lambda.powertools.tracing.Tracing;

import static software.amazon.lambda.powertools.tracing.CaptureMode.*;

/**
 * Handler for requests to Lambda function.
 */
public class App implements RequestHandler<APIGatewayProxyRequestEvent,
    APIGatewayProxyResponseEvent> {
    Logger log = LogManager.getLogger(App.class);

    @Logging(logEvent = true)
    @Tracing(captureMode = DISABLED)
    @Metrics(captureColdStart = true)
    public APIGatewayProxyResponseEvent handleRequest(final
    APIGatewayProxyRequestEvent input, final Context context) {
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        headers.put("X-Custom-Header", "application/json");
    }
}
```

```

        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent()
            .withHeaders(headers);
        try {
            final String pageContents = this.getPageContents("https://
checkip.amazonaws.com");
            String output = String.format("{ \"message\": \"hello world\",
\"location\": \"%s\" }", pageContents);

            return response
                .withStatusCode(200)
                .withBody(output);
        } catch (IOException e) {
            return response
                .withBody("{}")
                .withStatusCode(500);
        }
    }
    @Tracing(namespace = "getPageContents")
    private String getPageContents(String address) throws IOException {
        log.info("Retrieving {}", address);
        URL url = new URL(address);
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(url.openStream())))) {
            return br.lines().collect(Collectors.joining(System.lineSeparator()));
        }
    }
}

```

8. Abra `HelloWorldStack.java` desde el directorio `hello-world\src\main\java\com\myorg` y sustituya el código existente por el siguiente código. Este código utiliza el [Constructor de Lambda](#) y el [Constructor de ApiGatewayV2](#) para crear una API de REST y una función de Lambda.

```

package com.myorg;

import software.amazon.awscdk.*;
import software.amazon.awscdk.services.apigatewayv2.alpha.*;
import
    software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegration;
import
    software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegrationProps;
import software.amazon.awscdk.services.lambda.Code;

```

```
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.FunctionProps;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.lambda.Tracing;
import software.amazon.awscdk.services.logs.RetentionDays;
import software.amazon.awscdk.services.s3.assets.AssetOptions;
import software.constructs.Construct;

import java.util.Arrays;
import java.util.List;

import static java.util.Collections.singletonList;
import static software.amazon.awscdk.BundlingOutput.ARCHIVED;

public class HelloWorldStack extends Stack {
    public HelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloWorldStack(final Construct scope, final String id, final StackProps
props) {
        super(scope, id, props);

        List<String> functionPackagingInstructions = Arrays.asList(
            "/bin/sh",
            "-c",
            "cd Function " +
                "&& mvn clean install " +
                "&& cp /asset-input/Function/target/function.jar /asset-
output/"
        );
        BundlingOptions.Builder builderOptions = BundlingOptions.builder()
            .command(functionPackagingInstructions)
            .image(Runtime.JAVA_11.getBundlingImage())
            .volumes(singletonList(
                // Mount local .m2 repo to avoid download all the
dependencies again inside the container
                DockerVolume.builder()
                    .hostPath(System.getProperty("user.home") +
"/.m2/")
                    .containerPath("/root/.m2/")
                    .build()
            ))
            .user("root")
    }
}
```

```

        .outputType(ARCHIVED);

    Function function = new Function(this, "Function", FunctionProps.builder()
        .runtime(Runtime.JAVA_11)
        .code(Code.fromAsset("app", AssetOptions.builder()
            .bundling(builderOptions
                .command(functionPackagingInstructions)
                .build())
            .build()))
        .handler("helloworld.App::handleRequest")
        .memorySize(1024)
        .tracing(Tracing.ACTIVE)
        .timeout(Duration.seconds(10))
        .logRetention(RetentionDays.ONE_WEEK)
        .build());

    HttpApi httpApi = new HttpApi(this, "sample-api", HttpApiProps.builder()
        .apiName("sample-api")
        .build());

    httpApi.addRoutes(AddRoutesOptions.builder()
        .path("/")
        .methods(singletonList(HttpMethod.GET))
        .integration(new HttpLambdaIntegration("function", function,
HttpLambdaIntegrationProps.builder()
            .payloadFormatVersion(PayloadFormatVersion.VERSION_2_0)
            .build()))
        .build());

    new CfnOutput(this, "HttpApi", CfnOutputProps.builder()
        .description("Url for Http Api")
        .value(httpApi.getApiEndpoint())
        .build());
    }
}

```

- Abra `pom.xml` desde el directorio `hello-world` y sustituya el código existente por el siguiente código.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd"

```

```
    xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.myorg</groupId>
  <artifactId>hello-world</artifactId>
  <version>0.1</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <cdk.version>2.70.0</cdk.version>
    <constructs.version>[10.0.0,11.0.0)</constructs.version>
    <junit.version>5.7.1</junit.version>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>

      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <mainClass>com.myorg.HelloWorldApp</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <!-- AWS Cloud Development Kit -->
    <dependency>
      <groupId>software.amazon.awscdk</groupId>
      <artifactId>aws-cdk-lib</artifactId>
      <version>${cdk.version}</version>
```

```
    </dependency>
    <dependency>
      <groupId>software.constructs</groupId>
      <artifactId>constructs</artifactId>
      <version>${constructs.version}</version>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>software.amazon.awscdk</groupId>
      <artifactId>apigatewayv2-alpha</artifactId>
      <version>${cdk.version}-alpha.0</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.awscdk</groupId>
      <artifactId>apigatewayv2-integrations-alpha</artifactId>
      <version>${cdk.version}-alpha.0</version>
    </dependency>
  </dependencies>
</project>
```

10. Asegúrese de estar en el directorio `hello-world` e implemente la aplicación.

```
cdk deploy
```

11. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
'Stacks[0].Outputs[?OutputKey=='HttpApi'].OutputValue' --output text
```

12. Invoque el punto de conexión de la API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

### 13. Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYt0c9j -
  Edges: [1]
    Summary_statistics:
      - total requests: 1
      - ok count(2XX): 1
      - error count(4XX): 0
      - fault count(5XX): 0
      - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYt0c9j
  - Edges: []
    Summary_statistics:
      - total requests: 1
      - ok count(2XX): 1
      - error count(4XX): 0
      - fault count(5XX): 0
      - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYt0c9j - Edges: [0]
    Summary_statistics:
      - total requests: 0
      - ok count(2XX): 0
      - error count(4XX): 0
      - fault count(5XX): 0
      - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
(1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
- 0.924s - sam-app-HelloWorldFunction-YBg8yfYt0c9j [HTTP: 200]
- 0.016s - sam-app-HelloWorldFunction-YBg8yfYt0c9j
- 0.739s - Initialization
- 0.016s - Invocation
- 0.013s - ## lambda_handler
  - 0.000s - ## app.hello
- 0.000s - Overhead
```



14. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
cdk destroy
```

## Uso de ADOT para instrumentar las funciones de Java

ADOT proporciona [capas](#) de Lambda completamente administradas que empaquetan todo lo necesario para recopilar datos de telemetría mediante el OTel SDK. Utilizando esta capa, se pueden instrumentar las funciones de Lambda sin tener que modificar el código de ninguna función. También se puede configurar la capa para que realice una inicialización personalizada de OTel. Para obtener más información, consulte [Configuración personalizada del recopilador de ADOT en Lambda](#) en la documentación de ADOT.

Para los tiempos de ejecución de Java, puede elegir entre dos capas:

- Capa de Lambda administrada de AWS para ADOT Java (agente de instrumentación automática): esta capa transforma automáticamente el código de la función durante el inicio para recopilar datos de seguimiento. Para obtener instrucciones detalladas sobre cómo utilizar esta capa junto con el agente de ADOT Java, consulte [Soporte de Lambda de AWS Distro for OpenTelemetry para Java \(agente de instrumentación automática\)](#) en la documentación de ADOT.
- Capa de Lambda administrada de AWS para ADOT Java: esta capa también proporciona instrumentación integrada para las funciones de Lambda, pero requiere algunos cambios de código manuales para inicializar el OTel SDK. Para obtener instrucciones detalladas sobre cómo utilizar esta capa, consulte [Soporte de Lambda de AWS Distro for OpenTelemetry para Java](#) en la documentación de ADOT.

## Uso del X-Ray SDK para instrumentar las funciones de Java

Para registrar datos sobre las llamadas que realiza la función a otros recursos y servicios de la aplicación, se puede agregar el X-Ray SDK para Java a la configuración de compilación. En el siguiente ejemplo, se muestra una configuración de compilación de Gradle que incluye las bibliotecas que activan la instrumentación automática de los clientes de AWS SDK for Java 2.x.

Example [build.gradle](#): dependencias de seguimiento.

```
dependencies {
```

```
implementation platform('software.amazon.awssdk:bom:2.16.1')
implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')
...
implementation 'com.amazonaws:aws-xray-recorder-sdk-core'
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk'
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'
...
}
```

Una vez agregadas las dependencias correctas y realizados los cambios de código necesarios, active el seguimiento en la configuración de la función mediante la consola de Lambda o la API.

## Activación del seguimiento con la consola de Lambda

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

## Activación del seguimiento con la API de Lambda

Configure el rastreo en la función Lambda con AWS CLI o SDK de AWS, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Activación del seguimiento con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

## Interpretación de un seguimiento de X-Ray

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos

sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.



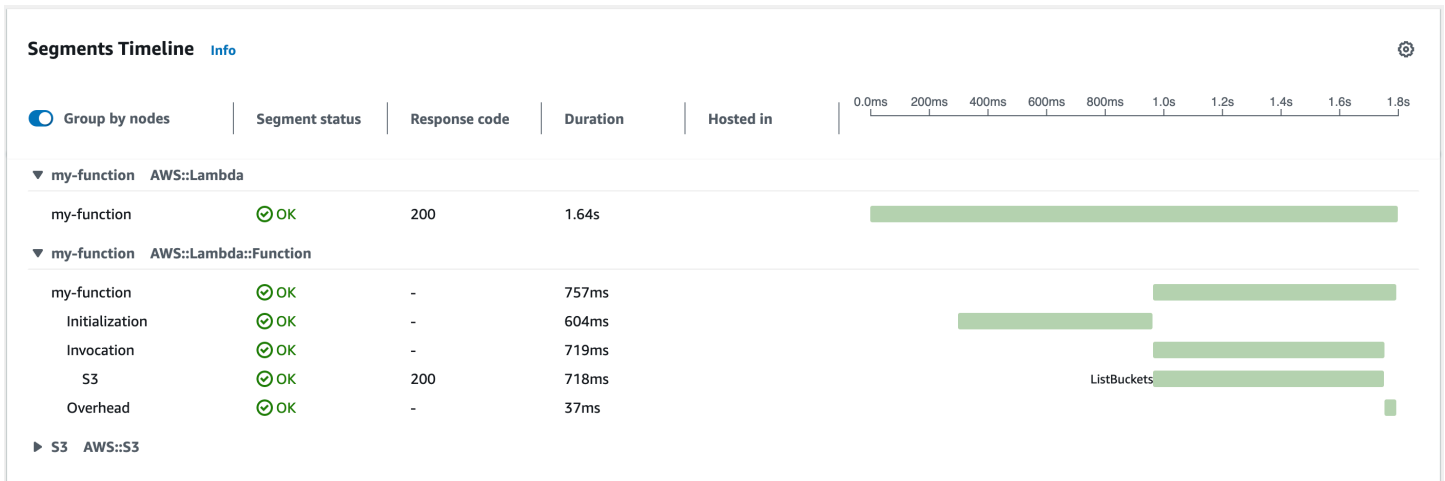
X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo,

se muestra un seguimiento con estos dos segmentos. Ambos se denominan `my-function`, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.

- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

#### Note

Las funciones de [Lambda SnapStart](#) también incluyen un subsegmento `Restore`. El subsegmento `Restore` muestra el tiempo que tarda Lambda en restaurar una instantánea, cargar el tiempo de ejecución (JVM) y ejecutar cualquier [enlace de tiempo de ejecución](#) de `afterRestore`. El proceso de restauración de instantáneas puede incluir el tiempo dedicado a actividades fuera de la micro VM. Esta vez se informa en el subsegmento `Restore`. No se le cobrará por el tiempo que pase fuera de la micro VM para restaurar una instantánea.

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte [AWS X-Ray SDK para Java](#) en la Guía para desarrolladores de AWS X-Ray.

#### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza cargos por almacenamiento y recuperación del seguimiento. Para más información, consulte [Precios de AWS X-Ray](#).

## Almacenamiento de dependencias de tiempo de ejecución en una capa (X-Ray SDK)

Si utiliza el X-Ray SDK para instrumentar el código de las funciones de los clientes del SDK de AWS, el paquete de implementación puede llegar a ser bastante grande. Para evitar que se carguen

dependencias en tiempo de ejecución cada vez que se actualice el código de las funciones, empaquete el X-Ray SDK en una [capa de Lambda](#).

En el siguiente ejemplo, se muestra un recurso de `AWS::Serverless::LayerVersion` que almacena el AWS SDK for Java y el X-Ray SDK.

Example [template.yml](#): capa de dependencias.

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: build/distributions/blank-java.zip
      Tracing: Active
      Layers:
        - !Ref libs
        ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-java-lib
      Description: Dependencies for the blank-java sample app.
      ContentUri: build/blank-java-lib.zip
      CompatibleRuntimes:
        - java21
```

Con esta configuración, solo se actualiza la capa de la biblioteca si se modifican las dependencias del tiempo de ejecución. Dado que el paquete de implementación de la función contiene únicamente el código, esto puede ayudar a reducir los tiempos de carga.

Para crear una capa de dependencias, es necesario realizar cambios en la configuración de compilación para generar el archivo de capas antes de la implementación. Para ver un ejemplo práctico, consulte la aplicación de ejemplo [java-basic](#) en GitHub.

## Seguimiento de X-Ray en aplicaciones de ejemplo (X-Ray SDK)

En el repositorio de GitHub correspondiente a esta guía hay aplicaciones de ejemplo que muestran cómo utilizar el seguimiento de X-Ray. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación y la limpieza, una plantilla de AWS SAM y recursos de soporte.

## Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

Todas las aplicaciones de ejemplo tienen activado el seguimiento activo de las funciones de Lambda. Por ejemplo, en la aplicación `s3-java`, se muestra la instrumentación automática de los clientes de AWS SDK for Java 2.x, la administración de segmentos para pruebas, subsegmentos personalizados y el uso de capas de Lambda para almacenar dependencias en tiempo de ejecución.



# Aplicaciones de ejemplo de Java para AWS Lambda

El repositorio de GitHub para esta guía ofrece aplicaciones de ejemplo en las que se muestra el uso de Java en AWS Lambda. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación y la limpieza, una plantilla de AWS CloudFormation y recursos de soporte.

## Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

## Ejecución de marcos Java populares en Lambda

- [spring-cloud-function-samples](#): un ejemplo de Spring que muestra cómo utilizar el marco [Spring Cloud Function](#) para crear funciones de Lambda AWS.
- [Demostración de la aplicación Spring Boot sin servidor](#): un ejemplo que muestra cómo configurar una aplicación Spring Boot típica en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de GraalVM con un tiempo de ejecución personalizado.
- [Demostración de la aplicación Micronaut sin servidor](#): un ejemplo que muestra cómo usar Micronaut en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de GraalVM con un tiempo de ejecución personalizado. Obtenga más información en las [guías de Micronaut/Lambda](#).
- [Demostración de la aplicación Quarkus sin servidor](#): un ejemplo que muestra cómo usar Quarkus en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de

GraalVM con un tiempo de ejecución personalizado. [Obtenga más información en la guía de Quarkus/Lambda y en la guía de Quarkus/SnapStart.](#)

Si es la primera vez que utiliza las funciones de Lambda en Java, comience con los ejemplos de `java-basic`. Para comenzar con los orígenes de eventos de Lambda, consulte los ejemplos de `java-events`. Ambos conjuntos de ejemplos muestran el uso de las bibliotecas Java de Lambda, las variables de entorno, el SDK de AWS y el SDK de AWS X-Ray. Estos ejemplos requieren una configuración mínima y puede implementarlos desde la línea de comandos en menos de un minuto.

# Creación de funciones de Lambda con Go

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice un [tiempo de ejecución exclusivo para el sistema operativo](#) (la familia de tiempos de ejecución provided) para implementar las funciones de Go en Lambda.

## Temas

- [Compatibilidad del tiempo de ejecución de Go](#)
- [Herramientas y bibliotecas](#)
- [Definir el controlador de funciones de Lambda en Go](#)
- [Uso del objeto de contexto Lambda para recuperar la información de la función Go](#)
- [Implementar funciones de Lambda en Go con archivos .zip](#)
- [Implemente funciones Go Lambda con imágenes de contenedor](#)
- [Registro y supervisión de las funciones de Lambda en Go](#)
- [Instrumentación del código Go en AWS Lambda](#)

## Compatibilidad del tiempo de ejecución de Go

El tiempo de ejecución administrado de Go 1.x para Lambda ha quedado [obsoleto](#). Si tiene funciones que usan el tiempo de ejecución de Go 1.x, debe migrar sus funciones a `provided.al2023` o `provided.al2`. Los tiempos de ejecución `provided.al2023` y `provided.al2` ofrecen varias ventajas sobre `go1.x`, incluida la compatibilidad con la arquitectura arm64 (procesadores Graviton2 de AWS), binarios más pequeños y tiempos de invocación ligeramente más rápidos.

No se requieren cambios de código para esta migración. Los únicos cambios necesarios se refieren a la forma en que se compila el paquete de implementación y al tiempo de ejecución que se utiliza para crear la función. Para obtener más información, consulte [Migración de las funciones de AWS Lambda del tiempo de ejecución de Go1.x al tiempo de ejecución personalizado en Amazon Linux 2](#) en el Blog de informática de AWS.

## Exclusivo del sistema operativo

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Tiempo de ejecución exclusivo del sistema operativo	provided. a12023	Amazon Linux 2023			
Tiempo de ejecución exclusivo del sistema operativo	provided. a12	Amazon Linux 2			

## Herramientas y bibliotecas

Lambda proporciona las siguientes herramientas y bibliotecas para el tiempo de ejecución Go:

- [AWS SDK para Go](#): el SDK oficial de AWS para el lenguaje de programación Go.
- [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/lambda): la implementación del modelo de programación de Lambda para Go. AWS Lambda utiliza este paquete para invocar su [controlador](#).
- [github.com/aws/aws-lambda-go/lambdacontext](https://github.com/aws/aws-lambda-go/lambdacontext): elementos auxiliares para obtener acceso a la información de contexto del [objeto de contexto](#).
- [github.com/aws/aws-lambda-go/events](https://github.com/aws/aws-lambda-go/events): esta biblioteca proporciona definiciones de tipos para las integraciones de orígenes de eventos comunes.
- [github.com/aws/aws-lambda-go/cmd/build-lambda-zip](https://github.com/aws/aws-lambda-go/cmd/build-lambda-zip): Esta herramienta se puede utilizar para crear un archivo.zip en Windows.

Para obtener más información, consulte [aws-lambda-go](#) en GitHub.

Lambda proporciona las siguientes aplicaciones de ejemplo para el tiempo de ejecución Go:

## Aplicaciones de ejemplo de Lambda en Go

- [go-al2](#): una función de “Hola, mundo” que devuelve la dirección IP pública. Esta aplicación utiliza el tiempo de ejecución personalizado `provided.al2`.
- [blank-go](#): una función Go que muestra el uso de las bibliotecas de Go de Lambda, el registro, las variables de entorno y el AWS SDK. Esta aplicación utiliza el tiempo de ejecución `go1.x`.

# Definir el controlador de funciones de Lambda en Go

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

## Temas

- [Conceptos básicos del controlador de Go](#)
- [Denominación](#)
- [Controlador de funciones de Lambda con tipos estructurados](#)
- [Uso del estado global](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Go](#)

## Conceptos básicos del controlador de Go

Una función de Lambda escrita en [Go](#) se crea como ejecutable de Go. En el código de la función de Lambda, debe incluir el paquete [github.com/aws/aws-lambda-go/lambda](https://github.com/aws/aws-lambda-go/lambda), que implementa el modelo de programación de para Go. Además, debe implementar el código de la función de controlador y una función de `main()`.

Example Función de Lambda de Go.

```
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(ctx context.Context, event *MyEvent) (*string, error) {
    if event == nil {
        return nil, fmt.Errorf("received nil event")
    }
}
```

```
message := fmt.Sprintf("Hello %s!", event.Name)
return &message, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

Aquí se muestra un ejemplo de entrada para esta función:

```
{
  "name": "Jane"
}
```

Tenga en cuenta lo siguiente:

- `package main`: en Go, el paquete que contiene `func main()` debe denominarse siempre `main`.
- `import`: utilice esta opción para incluir las bibliotecas que requiere la función de Lambda. En este ejemplo, incluye:
  - `context`: [Uso del objeto de contexto Lambda para recuperar la información de la función Go](#).
  - `fmt`: objeto de [formato](#) de Go que se utiliza para dar formato al valor de retorno de la función.
  - `github.com/aws/aws-lambda-go/lambda`: como se mencionó anteriormente, implementa el modelo de programación de Lambda para Go.
- `func HandleRequest(ctx context.Context, name MyEvent) (string, error)`: esta es la firma de controlador de Lambda. Es el punto de entrada de la función de Lambda y contiene la lógica que se ejecuta cuando se invoca la función. Además, los parámetros incluidos indican lo siguiente:
  - `ctx context.Context`: proporciona información en tiempo de ejecución para la invocación de la función de Lambda. `ctx` es la variable que declara para aprovechar la información disponible a través de [Uso del objeto de contexto Lambda para recuperar la información de la función Go](#).
  - `event *myEvent`: se trata de un parámetro de nombre `event` que apunta a `MyEvent`. Representa la entrada a la función de Lambda.
  - `*string, error`: el controlador devuelve dos valores. El primero es un puntero a una cadena que contiene el resultado de la función de Lambda. El segundo es un tipo de error, que es `nil` si no hay ningún error y contiene información de [error](#) estándar si algo sale mal.
  - `return &message, nil`: devuelve dos valores. El primero es un puntero a un mensaje de cadena, que es un saludo creado con el campo `Name` del evento de entrada. El segundo valor, `nil`, indica que la función no ha detectado ningún error.

- `func main()`: el punto de entrada que ejecuta el código de la función de Lambda. Esto es necesario.

Al agregar `lambda.Start(HandleRequest)` entre los corchetes del código `func main(){}`, la función de Lambda se ejecuta. De acuerdo con los estándares del lenguaje Go, el corchete de apertura `{` debe colocarse directamente al final de la firma de la función `main`.

## Denominación

tiempos de ejecución `provided.al2` y `provided.al2023`

Para las funciones de Go que utilizan el tiempo de ejecución `provided.al2` o `provided.al2023` en un [paquete de implementación .zip](#), el archivo ejecutable que contiene el código de la función debe tener el nombre `bootstrap`. Si va a implementar la función de con un archivo `.zip`, el archivo `bootstrap` debe estar en la raíz del archivo `.zip`. Para las funciones de Go que utilizan el tiempo de ejecución `provided.al2` o `provided.al2023` en una [imagen de contenedor](#), puede utilizar cualquier nombre para el archivo ejecutable.

Puede usar cualquier nombre para el controlador. Para hacer referencia al valor del controlador en su código, puede utilizar la variable de entorno `_HANDLER`.

Tiempo de ejecución `go1.x`

Para las funciones de Go que utilizan el tiempo de ejecución `go1.x`, el archivo ejecutable y el controlador pueden compartir cualquier nombre. Por ejemplo, si establece el valor del controlador como `Handler`, Lambda llamará a la función `main()` del archivo ejecutable `Handler`.

Para cambiar el nombre del controlador de funciones en la consola de Lambda en la que se muestra `Runtime settings` (Configuración de tiempo de ejecución), seleccione `Edit` (Editar).

## Controlador de funciones de Lambda con tipos estructurados

En el ejemplo anterior, el tipo de entrada era una cadena sencilla. Sin embargo, también pueden pasarse eventos estructurados al controlador de funciones:

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
```



```
)

type MyEvent struct {
    Name string `json:"What is your name?"`
    Age  int   `json:"How old are you?"`
}

type MyResponse struct {
    Message string `json:"Answer"`
}

func HandleLambdaEvent(event *MyEvent) (*MyResponse, error) {
    if event == nil {
        return nil, fmt.Errorf("received nil event")
    }
    return &MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,
        event.Age)}, nil
}

func main() {
    lambda.Start(HandleLambdaEvent)
}
```

Aquí se muestra un ejemplo de entrada para esta función:

```
{
  "What is your name?": "Jim",
  "How old are you?": 33
}
```

Una respuesta tiene un aspecto similar al siguiente:

```
{
  "Answer": "Jim is 33 years old!"
}
```

Para que se exporten, los nombres de campo en la estructura de eventos deben estar en mayúsculas. Para obtener más información acerca de cómo controlar eventos de orígenes de eventos de AWS, consulte [aws-lambda-go/events](https://aws.amazon.com/lambda-go/events/).

## Firmas de controlador válidas

Al crear un controlador de funciones de Lambda en Go tiene varias opciones, pero debe cumplir las reglas siguientes:

- El controlador debe ser una función.
- El controlador puede aceptar entre 0 y 2 argumentos. Si hay dos argumentos, el primero debe implementar `context.Context`.
- El controlador puede devolver entre 0 y 2 argumentos. Si hay un único valor de retorno, este debe implementar `error`. Si hay dos valores de retorno, el segundo debe implementar `error`.

A continuación se enumeran las firmas de controlador válidas. `TIn` y `TOut` representan los tipos compatibles con la biblioteca estándar `encoding/json`. Para obtener más información, consulte [func Unmarshal](#) a fin de saber cómo se deserializan estos tipos.

- `func ()`
- `func () error`
- `func (TIn) error`
- `func () (TOut, error)`
- `func (context.Context) error`
- `func (context.Context, TIn) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) (TOut, error)`

## Uso del estado global

Es posible declarar y modificar variables globales que son independientes del código de controlador de la función de Lambda. Además, el controlador puede declarar una función de `init` que se

ejecute cuando este se carga. El comportamiento es el mismo en AWS Lambda que en los programas Go estándar. Una sola instancia de su función de Lambda nunca controlará varios eventos de forma simultánea.

### Example Función de Go con variables globales

#### Note

Este código usa la AWS SDK for Go V2. Para obtener más información, consulte [Introducción a AWS SDK for Go V2](#).

```
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/aws-sdk-go-v2/service/s3/types"
    "log"
)

var invokeCount int
var myObjects []types.Object

func init() {
    // Load the SDK configuration
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalf("Unable to load SDK config: %v", err)
    }

    // Initialize an S3 client
    svc := s3.NewFromConfig(cfg)

    // Define the bucket name as a variable so we can take its address
    bucketName := "amzn-s3-demo-bucket"
    input := &s3.ListObjectsV2Input{
        Bucket: &bucketName,
    }
}
```

```
// List objects in the bucket
result, err := svc.ListObjectsV2(context.TODO(), input)
if err != nil {
    log.Fatalf("Failed to list objects: %v", err)
}
myObjects = result.Contents
}

func LambdaHandler(ctx context.Context) (int, error) {
    invokeCount++
    for i, obj := range myObjects {
        log.Printf("object[%d] size: %d key: %s", i, obj.Size, *obj.Key)
    }
    return invokeCount, nil
}

func main() {
    lambda.Start(LambdaHandler)
}
```

## Prácticas recomendadas de codificación para las funciones de Lambda en Go

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación.
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Uso del objeto de contexto Lambda para recuperar la información de la función Go

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona métodos y propiedades con información acerca de la invocación, la función y el entorno de ejecución.

La biblioteca de contexto de Lambda proporciona las siguientes variables globales, métodos y propiedades.

## Variables globales

- `FunctionName`: el nombre de la función de Lambda.
- `FunctionVersion`: la [versión](#) de la función.
- `MemoryLimitInMB`: cantidad de memoria asignada a la función.
- `LogGroupName`: el grupo de registros de para la función.
- `LogStreamName`: el flujo de registro de la instancia de la función.

## Métodos de context

- `Deadline`: devuelve la fecha en la que la ejecución agota su tiempo de espera, en milisegundos de tiempo Unix.

## Propiedades de context

- `InvokedFunctionArn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar la función. Indica si el invocador especificó un número de versión o alias.
- `AwsRequestId`: el identificador de la solicitud de invocación.
- `Identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
- `ClientContext`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.

## Acceso a la información del contexto de invocación

Las funciones de Lambda tienen acceso a metadatos acerca de su entorno y la solicitud de invocación. Se puede obtener acceso en el [contexto del paquete](#). Si el controlador incluye

`context.Context` como parámetro, Lambda información acerca de su función en la propiedad `Value` del contexto. Tenga en cuenta que debe importar la biblioteca `lambdacontext` para obtener acceso al contenido del objeto `context.Context`.

```
package main

import (
    "context"
    "log"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/lambdacontext"
)

func CognitoHandler(ctx context.Context) {
    lc, _ := lambdacontext.FromContext(ctx)
    log.Print(lc.Identity.CognitoIdentityPoolID)
}

func main() {
    lambda.Start(CognitoHandler)
}
```

En el ejemplo anterior, `lc` es la variable que se usa para consumir la información que el objeto `context` capturó y `log.Print(lc.Identity.CognitoIdentityPoolID)` imprime dicha información, en este caso, `CognitoIdentityPoolID`.

En el ejemplo siguiente se indica cómo utilizar el objeto `context` para monitorear el tiempo que tarda en ejecutarse la función de Lambda. Esto le permite analizar las expectativas de desempeño y ajustar el código de su función en consecuencia, si es necesario.

```
package main

import (
    "context"
    "log"
    "time"
    "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

    deadline, _ := ctx.Deadline()
```

```
deadline = deadline.Add(-100 * time.Millisecond)
timeoutChannel := time.After(time.Until(deadline))

for {

    select {

        case <- timeoutChannel:
            return "Finished before timing out.", nil

        default:
            log.Print("hello!")
            time.Sleep(50 * time.Millisecond)
    }
}

}

func main() {
    lambda.Start(LongRunningHandler)
}
```



# Implementar funciones de Lambda en Go con archivos .zip

El código de la función AWS Lambda se compone de scripts o programas compilados y sus dependencias. Utiliza un paquete de implementación para implementar su código de función en Lambda. Lambda admite dos tipos de paquetes de implementación: imágenes de contenedor y archivos .zip.

En esta página se describe cómo crear un archivo .zip como paquete de despliegue para el tiempo de ejecución de Go y, a continuación, utilizar el archivo .zip para implementar el código de función con AWS Lambda mediante la AWS Management Console, la AWS Command Line Interface (AWS CLI) y AWS Serverless Application Model (AWS SAM).

Lambda usa permisos de archivo de POSIX, por lo que puede necesitar [establecer permisos para la carpeta del paquete de despliegue](#) antes de crear el archivo .zip.

## Secciones

- [Creación de un archivo.zip en macOS y Linux](#)
- [Crear un archivo.zip en Windows](#)
- [Creación y actualización de funciones de Lambda en Go mediante archivos .zip](#)
- [Creación de una capa de Go para las dependencias](#)

## Creación de un archivo.zip en macOS y Linux

En los siguientes pasos, se muestra cómo compilar el archivo ejecutable mediante el comando `go build` y crear un paquete de implementación de archivos .zip para Lambda. Antes de compilar el código, asegúrese de haber instalado el paquete [lambda](#) de GitHub. Este módulo proporciona una implementación de la interfaz del tiempo de ejecución que administra la interacción entre Lambda y el código de la función. Para descargarla, ejecute el siguiente comando.

```
go get github.com/aws/aws-lambda-go/lambda
```

Si su función usa AWS SDK for Go, descargue el conjunto estándar de módulos del SDK, junto con los clientes de API de servicio de AWS que requiera su aplicación. Para obtener información sobre cómo instalar el SDK para Go, consulte [Getting Started with the AWS SDK for Go V2](#).

## Uso de la familia de tiempo de ejecución proporcionada

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice un [tiempo de ejecución exclusivo para el sistema operativo](#) (la familia de tiempos de ejecución `provided`) para implementar las funciones de Go en Lambda.

Para crear un paquete de implementación `.zip` (macOS o Linux)

1. En el directorio del proyecto que contiene el archivo `main.go` de la aplicación, compile el ejecutable. Tenga en cuenta lo siguiente:
  - El ejecutable debe denominarse `bootstrap`. Para obtener más información, consulte [Denominación](#).
  - Establezca su [arquitectura del conjunto de instrucciones](#) de destino. El tiempo de ejecución de OS solo admite `arm64` y `x86_64`.
  - Puede utilizar la etiqueta opcional `lambda.norpc` para excluir el componente de llamada a procedimiento remoto (RPC) de la biblioteca [lambda](#). El componente de RPC solo es necesario si utiliza el tiempo de ejecución obsoleto Go 1.x. La exclusión del RPC reduce el tamaño del paquete de implementación.

Para la arquitectura `arm64`:

```
GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
```

Para la arquitectura `x86_64`:

```
GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
```


2. (Opcional) Es posible que necesite compilar paquetes con `CGO_ENABLED=0` establecido en Linux:

```
GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc main.go
```

Este comando crea un paquete binario estable para las versiones estándar de la biblioteca de C (`libc`), que puede ser diferente en Lambda y en otros dispositivos.

3. Empaquete el ejecutable en un archivo `.zip` para crear un paquete de implementación.

```
zip myFunction.zip bootstrap
```

 Note

El archivo `bootstrap` debe estar en la raíz del archivo `zip`.


4. Cree la función. Tenga en cuenta lo siguiente:

- El binario debe tener un nombre `bootstrap`, pero el nombre del controlador puede ser cualquier cosa. Para obtener más información, consulte [Denominación](#).
- La opción `--architectures` es obligatoria si utiliza `arm64`. El valor predeterminado es `x86_64`.
- Para `--role`, especifique el Nombre de recurso de Amazon (ARN) del [rol de ejecución](#).

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures arm64 \  
--role arn:aws:iam::111122223333:role/lambda-ex \  
--zip-file fileb://myFunction.zip
```

## Crear un archivo.zip en Windows

En los siguientes pasos, se muestra cómo descargar la herramienta [build-lambda-zip](#) para Windows desde GitHub, compilar el archivo ejecutable y crear un paquete de despliegue `.zip`.

 Note

Si aún no lo ha hecho, debe instalar [git](#) y luego agregar el `git` ejecutable a su variable de entorno de Windows `%PATH%`.

Antes de compilar el código, asegúrese de haber instalado la biblioteca [lambda](#) desde GitHub. Para descargarla, ejecute el siguiente comando.

```
go get github.com/aws/aws-lambda-go/lambda
```

Si su función usa AWS SDK for Go, descargue el conjunto estándar de módulos del SDK, junto con los clientes de API de servicio de AWS que requiera su aplicación. Para obtener información sobre cómo instalar el SDK para Go, consulte [Getting Started with the AWS SDK for Go V2](#).

## Uso de la familia de tiempo de ejecución proporcionada

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice un [tiempo de ejecución exclusivo para el sistema operativo](#) (la familia de tiempos de ejecución provided) para implementar las funciones de Go en Lambda.

Para crear un paquete de implementación .zip (Windows)

1. Descargue la herramienta build-lambda-zip de GitHub.

```
go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
```

2. Utilice la herramienta desde GOPATH para crear un archivo .zip. Si tiene una instalación predeterminada de Go, la herramienta suele estar en %USERPROFILE%\Go\bin. De lo contrario, vaya hasta la ubicación donde instaló el tiempo de ejecución de Go y haga una de las siguientes acciones:

cmd.exe

En cmd.exe, ejecute una de las siguientes opciones, según su [arquitectura del conjunto de instrucciones](#) de destino. El tiempo de ejecución de OS solo admite arm64 y x86\_64.

Puede utilizar la etiqueta opcional `lambda.norpc` para excluir el componente de llamada a procedimiento remoto (RPC) de la biblioteca [lambda](#). El componente de RPC solo es necesario si utiliza el tiempo de ejecución obsoleto Go 1.x. La exclusión del RPC reduce el tamaño del paquete de implementación.

Example — Para la arquitectura x86\_64

```
set GOOS=linux
set GOARCH=amd64
set CGO_ENABLED=0
go build -tags lambda.norpc -o bootstrap main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

## Example — Para la arquitectura arm64

```
set GOOS=linux
set GOARCH=arm64
set CGO_ENABLED=0
go build -tags lambda.norpc -o bootstrap main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

## PowerShell

En PowerShell, ejecute una de las siguientes opciones, según su [arquitectura del conjunto de instrucciones](#) de destino. El tiempo de ejecución de SO solo admite arm64 y x86\_64.

Puede utilizar la etiqueta opcional `lambda.norpc` para excluir el componente de llamada a procedimiento remoto (RPC) de la biblioteca [lambda](#). El componente de RPC solo es necesario si utiliza el tiempo de ejecución obsoleto Go 1.x. La exclusión del RPC reduce el tamaño del paquete de implementación.

Para la arquitectura x86\_64:

```
$env:GOOS = "linux"
$env:GOARCH = "amd64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

Para la arquitectura arm64:

```
$env:GOOS = "linux"
$env:GOARCH = "arm64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

### 3. Cree la función. Tenga en cuenta lo siguiente:

- El binario debe tener un nombre `bootstrap`, pero el nombre del controlador puede ser cualquier cosa. Para obtener más información, consulte [Denominación](#).

- La opción `--architectures` es obligatoria si utiliza `arm64`. El valor predeterminado es `x86_64`.
- Para `--role`, especifique el Nombre de recurso de Amazon (ARN) del [rol de ejecución](#).

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures arm64 \  
--role arn:aws:iam::111122223333:role/Lambda-ex \  
--zip-file fileb://myFunction.zip
```

## Creación y actualización de funciones de Lambda en Go mediante archivos .zip

Una vez que haya creado su paquete de despliegue .zip, puede usarlo para crear una nueva función de Lambda o actualizar una existente. Puede implementar el paquete .zip a través de la consola, la AWS Command Line Interface y la API de Lambda. También puede crear y actualizar funciones de Lambda mediante AWS Serverless Application Model (AWS SAM) y AWS CloudFormation.

El tamaño máximo de un paquete de despliegue .zip para Lambda es de 250 MB (descomprimido). Tenga en cuenta que este límite se aplica al tamaño combinado de todos los archivos que cargue, incluidas las capas de Lambda.

El tiempo de ejecución de Lambda necesita permiso para leer los archivos del paquete de implementación. En la notación octal de permisos de Linux, Lambda necesita 644 permisos para archivos no ejecutables (`rw-r--r--`) y 755 permisos (`rw-r-xr-x`) para directorios y archivos ejecutables.

En Linux y macOS, utilice el comando `chmod` para cambiar los permisos de los archivos y directorios del paquete de implementación. Por ejemplo, para brindarle a un archivo ejecutable los permisos correctos, ejecute el siguiente comando.

```
chmod 755 <filepath>
```

Para cambiar los permisos de los archivos en Windows, consulte [Set, View, Change, or Remove Permissions on an Object](#) en la documentación de Microsoft Windows.

## Creación y actualización de funciones con archivos .zip mediante la consola

Para crear una nueva función, primero debe crearla en la consola y, a continuación, cargar el archivo .zip. Para actualizar una función existente, abra la página de la función correspondiente y, a continuación, siga el mismo procedimiento para agregar el archivo .zip actualizado.

Si el archivo .zip tiene un tamaño inferior a 50 MB, puede crear o actualizar una función al cargarlo directamente desde su equipo local. Para archivos .zip de más de 50 MB, primero debe cargar su paquete en un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS Management Console, consulte [Introducción a Amazon S3](#). Para cargar archivos mediante la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

### Note

No se puede convertir una función de imagen de contenedor existente para utilizar un archivo .zip. Debe crear una nueva función.

Para crear una nueva función (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija Crear función.
2. Elija Crear desde cero.
3. En Información básica, haga lo siguiente:
  - a. En Nombre de función, escriba el nombre de la función.
  - b. En Runtime (Tiempo de ejecución), elija `provided.al2023`.
4. (Opcional) En Permisos, expanda Cambiar función de ejecución predeterminada. Puede crear un nuevo Rol de ejecución o utilizar uno existente.
5. Elija Crear función. Lambda crea una función básica “Hola, mundo” mediante el tiempo de ejecución elegido.

Para cargar un archivo .zip desde su equipo local (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar el archivo .zip.
2. Seleccione la pestaña Código.

3. En el panel Código fuente, elija Cargar desde.
4. Elija un archivo .zip.
5. Para cargar el archivo .zip, haga lo siguiente:
  - a. Seleccione Cargar y, a continuación, seleccione su archivo .zip en el selector de archivos.
  - b. Elija Abrir.
  - c. Seleccione Guardar.

#### Carga de un archivo .zip desde un bucket de Amazon S3 (consola)

1. En la [página Funciones](#) de la consola de Lambda, elija la función para la que desea cargar un nuevo archivo .zip.
2. Seleccione la pestaña Código.
3. En el panel Código fuente, elija Cargar desde.
4. Elija la ubicación de Amazon S3.
5. Pegue la URL del enlace de Amazon S3 de su archivo .zip y seleccione Guardar.

#### Creación y actualización de funciones con archivos .zip mediante la AWS CLI

Puede utilizar la [AWS CLI](#) para crear una nueva función o actualizar una existente con un archivo .zip. Utilice los comandos [create-function](#) y [update-function-code](#) para implementar su paquete .zip. Si el archivo .zip tiene un tamaño inferior a 50 MB, puede cargarlo desde una ubicación de archivo en su equipo de compilación local. Para archivos más grandes, debe cargar su paquete .zip desde un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

#### Note

Si carga su archivo .zip desde un bucket de Amazon S3 con la AWS CLI, el bucket debe estar ubicado en la misma Región de AWS que su función.

Para crear una nueva función mediante un archivo .zip con la AWS CLI, debe especificar lo siguiente:

- El nombre de la función (`--function-name`).



- El tiempo de ejecución de la función (`--runtime`).
- El nombre de recurso de Amazon (ARN) del [rol de ejecución](#) de la función (`--role`).
- El nombre del método de controlador en el código de la función (`--handler`).

También debe especificar la ubicación del archivo `.zip`. Si el archivo `.zip` se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo `.zip` en un bucket de Amazon S3, utilice la opción `--code`, como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `S3ObjectVersion` para los objetos con versiones.

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \  
--code S3Bucket=amzn-s3-demo-  
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Para actualizar una función existente mediante la CLI, especifique el nombre de la función mediante el parámetro `--function-name`. También debe especificar la ubicación del archivo `.zip` que desea utilizar para actualizar el código de la función. Si el archivo `.zip` se encuentra en una carpeta de su equipo de compilación local, utilice la opción `--zip-file` para especificar la ruta del archivo, como se muestra en el siguiente comando de ejemplo.

```
aws lambda update-function-code --function-name myFunction \  
--zip-file fileb://myFunction.zip
```

Para especificar la ubicación del archivo `.zip` en un bucket de Amazon S3, utilice las opciones `--s3-bucket` y `--s3-key` tal como se muestra en el siguiente comando de ejemplo. Solo necesita utilizar el parámetro `--s3-object-version` para los objetos con versiones.

```
aws lambda update-function-code --function-name myFunction \  
--s3-bucket myBucket --s3-key myFileName.zip --s3-object-version myObjectVersion
```

```
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObjectVersion
```

## Creación y actualización de funciones con archivos .zip mediante la API de Lambda

Para crear y actualizar funciones con un archivo de archivos .zip, utilice las siguientes operaciones de la API:

- [CreateFunction](#)
- [UpdateFunctionCode](#)

## Creación y actualización de funciones con archivos .zip mediante AWS SAM

AWS Serverless Application Model (AWS SAM) es un conjunto de herramientas que ayuda a agilizar el proceso de creación y ejecución de aplicaciones sin servidor en AWS. Defina los recursos de su aplicación en una plantilla YAML o JSON y utilice la interfaz de la línea de comandos de AWS SAM (AWS SAM CLI) para crear, empaquetar e implementar sus aplicaciones. Al crear una función de Lambda a partir de una plantilla de AWS SAM, AWS SAM crea automáticamente un paquete de despliegue .zip o una imagen de contenedor con el código de la función y las dependencias que especifique. Para obtener más información sobre el uso de AWS SAM para crear e implementar funciones de Lambda, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

También puede utilizar AWS SAM para crear una función de Lambda con un archivo de archivos .zip existente. Para crear una función de Lambda mediante AWS SAM, puede guardar el archivo .zip en un bucket de Amazon S3 o en una carpeta local de su equipo de compilación. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS SAM, el recurso `AWS::Serverless::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `CodeUri`: se establece como el URI de Amazon S3, la ruta a la carpeta local o el objeto [FunctionCode](#) del código de la función.
- `Runtime`: se establece como el entorno de ejecución elegido

Con AWS SAM, si su archivo .zip tiene más de 50 MB, no es necesario cargarlo primero en un bucket de Amazon S3. AWS SAM puede cargar paquetes .zip hasta el tamaño máximo permitido de 250 MB (descomprimidos) desde una ubicación de su equipo de compilación local.

Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS SAM.

Ejemplo: uso de AWS SAM para crear una función de Go con provided.al2023

1. Cree una plantilla de AWS SAM con las propiedades siguientes:
  - BuildMethod: especifica el compilador de la aplicación. Utilice go1.x.
  - Tiempo de ejecución: use provided.al2023.
  - CodeUri: ingrese la ruta de su código.
  - Arquitecturas: use [arm64] para la arquitectura arm64. Use [amd64] o elimine la propiedad Architectures para la arquitectura del conjunto de instrucciones x86\_64.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: go1.x
    Properties:
      CodeUri: hello-world/ # folder where your main program resides
      Handler: bootstrap
      Runtime: provided.al2023
      Architectures: [arm64]
```

2. Utilice el comando [sam build](#) para compilar el ejecutable.

```
sam build
```

3. Utilice el comando [sam deploy](#) para implementar la función en Lambda.

```
sam deploy --guided
```

## Creación y actualización de funciones con archivos .zip mediante AWS CloudFormation

Puede utilizar AWS CloudFormation para crear una función de Lambda con un archivo de archivos .zip. Para crear una función de Lambda a partir de un archivo .zip, primero debe cargar el archivo a un bucket de Amazon S3. Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3 con la AWS CLI, consulte [Mover objetos](#) en la Guía del usuario de la AWS CLI.

En la plantilla de AWS CloudFormation, el recurso `AWS::Lambda::Function` especifica la función de Lambda. En este recurso, establezca las siguientes propiedades para crear una función mediante un archivo de archivos .zip:

- `PackageType`: se establece como `Zip`.
- `Code`: ingrese el nombre del bucket de Amazon S3 y el nombre del archivo .zip en los campos `S3Bucket` y `S3Key`.
- `Runtime`: se establece como el tiempo de ejecución elegido.

El archivo .zip que genera AWS CloudFormation no puede superar los 4 MB. Para obtener más información sobre la implementación de funciones mediante un archivo .zip en AWS CloudFormation, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

## Creación de una capa de Go para las dependencias

### Note

El uso de capas con funciones en un lenguaje compilado, como Go, puede no ofrecer las mismas ventajas que con un lenguaje interpretado, como Python. Como Go es un lenguaje compilado, las funciones aún tienen que cargar de forma manual los ensamblajes compartidos en la memoria durante la fase de inicio, lo que puede aumentar los tiempos de arranque en frío. En su lugar, se recomienda incluir cualquier código compartido en el momento de la compilación para aprovechar las optimizaciones del compilador integradas.

En las instrucciones de esta sección, se muestra cómo incluir las dependencias en una capa.

Lambda detecta de forma automática cualquier biblioteca en el directorio `/opt/lib` y todos los archivos binarios en el directorio `/opt/bin`. Para asegurarse de que Lambda encuentre el contenido de la capa de forma correcta, cree una capa con la siguiente estructura:

```
custom-layer.zip
# lib
  | lib_1
  | lib_2
# bin
  | bin_1
  | bin_2
```

Después de empaquetar la capa, consulte [the section called “Creación y eliminación de capas”](#) y [the section called “Adición de capas”](#) para completar la configuración de la capa.

# Implemente funciones Go Lambda con imágenes de contenedor


Hay dos formas de compilar una imagen de contenedor para una función de Lambda en Go:

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice una [imagen base exclusiva para el sistema operativo](#) para crear imágenes de Go para Lambda. Para que la imagen sea compatible con Lambda, debe incluir el paquete `aws-lambda-go/lambda` en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el paquete `aws-lambda-go/lambda` en la imagen.

 Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.

## Imágenes base de AWS para implementar funciones de Go

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice una [imagen base exclusiva para el sistema operativo](#) para implementar funciones de Go en Lambda.

## Exclusivo del sistema operativo

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
Tiempo de ejecución exclusivo del sistema operativo	provided.a12023	Amazon Linux 2023			
Tiempo de ejecución exclusivo del sistema operativo	provided.a12	Amazon Linux 2			

Galería pública de Amazon Elastic Container Registry: [gallery.ecr.aws/lambda/provided](https://gallery.ecr.aws/lambda/provided)

## Cliente de interfaz de tiempo de ejecución de Go

El paquete `aws-lambda-go/lambda` incluye una implementación de la interfaz de tiempo de ejecución. Para ver ejemplos de cómo usar `aws-lambda-go/lambda` en la imagen, consulte [Uso de una imagen base exclusiva del sistema operativo de AWS](#) o [Uso de una imagen base que no sea de AWS](#).

## Uso de una imagen base exclusiva del sistema operativo de AWS

Go se implementa de manera distinta a otros tiempos de ejecución administrados. Como Go se compila de forma nativa en un archivo binario ejecutable, no requiere un tiempo de ejecución de lenguaje dedicado. Utilice una [imagen base exclusiva para el sistema operativo](#) para compilar imágenes de contenedor para funciones de Go.

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
al2023	Tiempo de ejecución exclusivo del sistema operativo	Amazon Linux 2023	<a href="#">Dockerfile para tiempo de ejecución exclusivo del sistema operativo en GitHub</a>	
al2	Tiempo de ejecución exclusivo del sistema operativo	Amazon Linux 2	<a href="#">Dockerfile para tiempo de ejecución exclusivo del sistema operativo en GitHub</a>	

Para obtener más información acerca de estas imágenes base, consulte [provided](#) en la galería pública de Amazon ECR.

Debe incluir el paquete [aws-lambda-go/lambda](#) con su controlador de Go. Este paquete implementa el modelo de programación para Go, incluida la interfaz de tiempo de ejecución.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Go
- [Docker](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

Creación de una imagen a partir de la imagen base `provided.al2023`

Para compilar e implementar una función de Go con la imagen base de **provided.al2023**

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir hello
cd hello
```



## 2. Inicialice un nuevo módulo de Go.

```
go mod init example.com/hello-world
```

## 3. Agregue la biblioteca lambda como dependencia de su nuevo módulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

## 4. Cree un archivo con el nombre `main.go` y, a continuación, ábralo en un editor de texto. Se trata del código de la función de Lambda. Puede utilizar el siguiente código de muestra para realizar pruebas o sustituirlo por su propio código.

```
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)


func handler(ctx context.Context, event events.APIGatewayProxyRequest)
    (events.APIGatewayProxyResponse, error) {
    response := events.APIGatewayProxyResponse{
        StatusCode: 200,
        Body:       "\"Hello from Lambda!\"",
    }
    return response, nil
}

func main() {
    lambda.Start(handler)
}
```

## 5. Utilice un editor de texto para crear un Dockerfile en el directorio del proyecto. El siguiente Dockerfile de ejemplo utiliza una [compilación de varias etapas](#). Esto le permite utilizar una imagen base diferente en cada paso. Puede usar una imagen, como una [imagen base de Go](#), para compilar el código y crear el binario ejecutable. Luego puede utilizar una imagen diferente, como `provided.al2023`, en la instrucción final FROM para definir la imagen que va a implementar en Lambda. El proceso de compilación está separado de la imagen de implementación final, por lo que la imagen final solo contiene los archivos necesarios para ejecutar la aplicación.

Puede utilizar la etiqueta opcional `lambda.norpc` para excluir el componente de llamada a procedimiento remoto (RPC) de la biblioteca [lambda](#). El componente de RPC solo es necesario si utiliza el tiempo de ejecución obsoleto Go 1.x. La exclusión del RPC reduce el tamaño del paquete de implementación.

Example — Dockerfile de compilación de varias etapas


 Note

Asegúrese de que la versión de Go que especifique en su Dockerfile (por ejemplo, `golang:1.20`) sea la misma versión de Go que utilizó para crear su aplicación.

```
FROM golang:1.20 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build with optional lambda.norpc tag
COPY main.go .
RUN go build -tags lambda.norpc -o main main.go
# Copy artifacts to a clean image
FROM public.ecr.aws/lambda/provided:al2023
COPY --from=build /helloworld/main ./main
ENTRYPOINT [ "./main" ]
```

6. Compile la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

 Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

## (Opcional) Prueba de la imagen de forma local

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. El emulador de interfaz de tiempo de ejecución se incluye en la imagen base de `provided.al2023`.

Para ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:
  - `docker-image` es el nombre de la imagen y `test` es la etiqueta.
  - `./main` es el ENTRYPOINT de su Dockerfile.

```
docker run -d -p 9000:8080 \  
--entrypoint /usr/local/bin/aws-lambda-rie \  
docker-image:test ./main
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

2. Desde una nueva ventana de terminal, publique un evento en el siguiente punto de conexión mediante un comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Es posible que algunas funciones requieran una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d \  
'{"payload":"hello world!"}'
```

3. Obtenga el ID del contenedor.

```
docker ps
```

4. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace `3766c4ab331c` por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace `111122223333` por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    }
  }
}
```

```

    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

```

- Copie el valor de `repositoryUri` de la salida del paso anterior.
- Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

- Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

- [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
- Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

**Note**

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

**8. Invoque la función.**

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

**9. Para ver la salida de la función, compruebe el archivo `response.json`.**

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

## Uso de una imagen base que no sea de AWS

Puede crear una imagen de contenedor para Go a partir de una imagen base que no es de AWS. El Dockerfile de ejemplo en los siguientes pasos usa una [imagen base de Alpine](#).

Debe incluir el paquete [aws-lambda-go/lambda](#) con su controlador de Go. Este paquete implementa el modelo de programación para Go, incluida la interfaz de tiempo de ejecución.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- Go
- [Docker](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

## Creación de imágenes a partir de una imagen base alternativa

Para crear e implementar una función de Go con una imagen base de Alpine

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir hello
cd hello
```

2. Inicialice un nuevo módulo de Go.

```
go mod init example.com/hello-world
```

3. Agregue la biblioteca lambda como dependencia de su nuevo módulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Cree un archivo con el nombre `main.go` y, a continuación, ábralo en un editor de texto. Se trata del código de la función de Lambda. Puede utilizar el siguiente código de muestra para realizar pruebas o sustituirlo por su propio código.

```
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
(event events.APIGatewayProxyResponse, error) {
    response := events.APIGatewayProxyResponse{
        StatusCode: 200,
```

```

    Body:      "\"Hello from Lambda!\"",
  }
  return response, nil
}

func main() {
  lambda.Start(handler)
}

```

- Utilice un editor de texto para crear un Dockerfile en el directorio del proyecto. El siguiente Dockerfile de ejemplo usa una [imagen base de Alpine](#).

### Example Dockerfile

#### Note

Asegúrese de que la versión de Go que especifique en su Dockerfile (por ejemplo, `golang:1.20`) sea la misma versión de Go que utilizó para crear su aplicación.

```

FROM golang:1.20.2-alpine3.16 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
# Build
COPY main.go .
RUN go build -o main main.go
# Copy artifacts to a clean image
FROM alpine:3.16
COPY --from=build /helloworld/main /main
ENTRYPOINT [ "/main" ]

```

- Compile la imagen de Docker con el comando [docker build](#). En el siguiente ejemplo se asigna un nombre a la imagen `docker-image` y se le asigna la [etiqueta](#) `test`.

```
docker build --platform linux/amd64 -t docker-image:test .
```

#### Note

El comando especifica la opción `--platform linux/amd64` para garantizar que el contenedor sea compatible con el entorno de ejecución de Lambda, independientemente



de la arquitectura de la máquina de compilación. Si tiene intención de crear una función de Lambda con la arquitectura del conjunto de instrucciones ARM64, asegúrese de cambiar el comando para utilizar la opción `--platform linux/arm64` en su lugar.

(Opcional) Prueba de la imagen de forma local

Utilice el [emulador de interfaz de tiempo de ejecución](#) para probar la imagen localmente. Puede [crear el emulador en su imagen](#) o usar el procedimiento siguiente para instalarlo en su equipo local.

Para instalar y ejecutar el emulador de interfaz de tiempo de ejecución en su equipo local

1. Desde el directorio del proyecto, ejecute el siguiente comando para descargar el emulador de interfaz de tiempo de ejecución (arquitectura x86-64) de GitHub e instalarlo en su equipo local.

Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \  
  curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-  
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \  
  chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Para instalar el emulador arm64, reemplace la URL del repositorio de GitHub en el comando anterior por lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie-arm64
```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"  
if (-not (Test-Path $dirPath)) {  
  New-Item -Path $dirPath -ItemType Directory  
}  
  
$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/  
releases/latest/download/aws-lambda-rie"  
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"  
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Para instalar el emulador arm64, reemplace el `$downloadLink` con lo siguiente:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
```

2. Inicie la imagen de Docker con el comando `docker run`. Tenga en cuenta lo siguiente:

- `docker-image` es el nombre de la imagen y `test` es la etiqueta.
- `/main` es el ENTRYPOINT de su Dockerfile.

### Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
9000:8080 \
  --entrypoint /aws-lambda/aws-lambda-rie \
  docker-image:test \
  /main
```

### PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
/main
```

Este comando ejecuta la imagen como un contenedor y crea un punto de conexión local en `localhost:9000/2015-03-31/functions/function/invocations`.

#### Note

Si creó la imagen de Docker para la arquitectura del conjunto de instrucciones ARM64, asegúrese de utilizar la opción `--platform linux/arm64` en lugar de `--platform linux/amd64`.

3. Publique un evento en el punto de conexión local.

## Linux/macOS

En Linux y macOS, ejecute el siguiente comando `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d  
'{"payload":"hello world!"}'
```

## PowerShell

En PowerShell, ejecute el siguiente comando `Invoke-WebRequest`:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Este comando invoca la función con un evento vacío y devuelve una respuesta. Si utiliza su propio código de función en lugar del código de función de ejemplo, quizás quiera invocar la función con una carga JSON. Ejemplo:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/  
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType  
"application/json"
```

4. Obtenga el ID del contenedor.

```
docker ps
```

5. Use el comando [docker kill](#) para detener el contenedor. En este comando, reemplace `3766c4ab331c` por el ID del contenedor del paso anterior.

```
docker kill 3766c4ab331c
```

## Implementación de la imagen

Para cargar la imagen en Amazon ECR y crear la función de Lambda

1. Para autenticar la CLI de Docker en su registro de Amazon ECR, ejecute el comando [get-login-password](#).
  - Establezca el valor de `--region` en la Región de AWS en la que desee crear el repositorio de Amazon ECR.
  - Reemplace `111122223333` por el ID de su Cuenta de AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Cree un repositorio en Amazon ECR con el comando [create-repository](#).

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

### Note

El repositorio de Amazon ECR debe estar en la misma Región de AWS que la función de Lambda.

Si se realiza de la forma correcta, verá una respuesta como la siguiente:

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world",
    "registryId": "111122223333",
    "repositoryName": "hello-world",
    "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world",
    "createdAt": "2023-03-09T10:39:01+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": true
    }
  }
}
```

```

    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}

```

3. Copie el valor de `repositoryUri` de la salida del paso anterior.
4. Ejecute el comando [docker tag](#) para etiquetar la imagen local en su repositorio de Amazon ECR como la versión más reciente. En este comando:
  - Reemplace `docker-image:test` por el nombre y la [etiqueta](#) de su imagen de Docker.
  - Reemplace `<ECRrepositoryUri>` por el `repositoryUri` que ha copiado. Asegúrese de incluir `:latest` al final del URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Ejemplo:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Ejecute el comando [docker push](#) para implementar la imagen local en el repositorio de Amazon ECR. Asegúrese de incluir `:latest` al final del URI del repositorio.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. [Cree un rol de ejecución](#) para la función si aún no tiene uno. Necesitará el nombre de recurso de Amazon (ARN) del rol en el paso siguiente.
7. Cree la función de Lambda. En `ImageUri`, especifique el URI del repositorio anterior. Asegúrese de incluir `:latest` al final del URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

**Note**

Puede crear una función con una imagen de una cuenta AWS diferente, siempre que la imagen esté en la misma región que la función de Lambda. Para obtener más información, consulte [Permisos entre cuentas de Amazon ECR](#).

**8. Invoque la función.**

```
aws lambda invoke --function-name hello-world response.json
```

Debería ver una respuesta como la siguiente:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

**9. Para ver la salida de la función, compruebe el archivo `response.json`.**

Para actualizar el código de la función, debe volver a compilar la imagen, cargar la nueva imagen en el repositorio de Amazon ECR y, a continuación, utilizar el comando [update-function-code](#) para implementar la imagen en la función de Lambda.

Lambda resuelve la etiqueta de la imagen en un resumen de imagen específico. Esto significa que si apunta la etiqueta de imagen que se utilizó para implementar la función a una nueva imagen en Amazon ECR, Lambda no actualiza automáticamente la función para usar la nueva imagen. Para implementar la nueva imagen en la misma función de Lambda, debe usar el comando `update-function-code`, incluso si la etiqueta de la imagen en Amazon ECR sigue siendo la misma.

# Registro y supervisión de las funciones de Lambda en Go

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre y envía registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación al flujo de registro y retransmite los registros y otras salidas desde el código de la función. Para obtener más información, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Esta página describe cómo producir resultados de registro a partir del código de la función de Lambda o registros de acceso mediante AWS Command Line Interface, la consola de Lambda o la consola de CloudWatch.

## Secciones

- [Crear una función que devuelve registros](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)

## Crear una función que devuelve registros

Para generar registros desde el código de la función, puede utilizar los métodos del [paquete fmt](#) o cualquier biblioteca de registro que escriba en `stdout` o en `stderr`. En el ejemplo siguiente, se utiliza el [paquete de registro](#).

Example [main.go](#): registro

```
func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
    // event
    eventJson, _ := json.MarshalIndent(event, "", " ")
    log.Printf("EVENT: %s", eventJson)
    // environment variables
    log.Printf("REGION: %s", os.Getenv("AWS_REGION"))
    log.Println("ALL ENV VARS:")
    for _, element := range os.Environ() {
        log.Println(element)
    }
}
```

## Example formato de registro

```

START RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Version: $LATEST
2020/03/27 03:40:05 EVENT: {
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "md5fBody": "7b27xmplb47ff90a553787216d55d91d",
      "md5fMessageAttributes": "",
      "attributes": {
        "ApproximateFirstReceiveTimestamp": "1523232000001",
        "ApproximateReceiveCount": "1",
        "SenderId": "123456789012",
        "SentTimestamp": "1523232000000"
      }
    },
    ...
  ]
}
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMPL6XBC
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$LATEST
2020/03/27 03:40:05 AWS_EXECUTION_ENV=AWS_Lambda_go1.x
END RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71
REPORT RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Duration: 38.66 ms Billed
Duration: 39 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled:
true

```

El tiempo de ejecución de Go registra las líneas START, END y REPORT de cada invocación. La línea del informe proporciona los siguientes detalles.

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.



- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- Traceld de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRi0C1mMTU0LTExZTgt0GNkYS0y0Tc0YzVlNGZiMjEgVmVyc2l1vb...",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad `base64` está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar `base64 -D`.

Example `get-logs.sh` script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza `sed` para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando `get-log-events`.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como `get-logs.sh`.

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

## Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
```

```
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

# Instrumentación del código Go en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas dos bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK for Go](#): un SDK para generar y enviar datos de seguimiento a X-Ray.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de ADOT para instrumentar las funciones de Go](#)
- [Uso del SDK de X-Ray para instrumentar las funciones de Go](#)
- [Activación del seguimiento con la consola de Lambda](#)
- [Activación del seguimiento con la API de Lambda](#)
- [Activación del seguimiento con AWS CloudFormation](#)
- [Interpretación de un seguimiento de X-Ray](#)

## Uso de ADOT para instrumentar las funciones de Go

ADOT proporciona [capas](#) de Lambda completamente administradas que empaquetan todo lo necesario para recopilar datos de telemetría mediante el OTel SDK. Utilizando esta capa, se pueden instrumentar las funciones de Lambda sin tener que modificar el código de ninguna función. También se puede configurar la capa para que realice una inicialización personalizada de OTel. Para obtener más información, consulte [Configuración personalizada del recopilador de ADOT en Lambda](#) en la documentación de ADOT.

Para los tiempos de ejecución de Go, puede agregar la capa de Lambda administrada por AWS para ADOT Go a fin de instrumentar de forma automática las funciones. Para obtener instrucciones detalladas sobre cómo agregar esta capa, consulte [Soporte de Lambda de AWS Distro OpenTelemetry para Go](#) en la documentación de ADOT.

## Uso del SDK de X-Ray para instrumentar las funciones de Go

Si desea registrar detalles sobre las llamadas que realiza la función de Lambda a otros recursos de la aplicación, también puede utilizar el SDK de AWS X-Ray para Go. Para obtener el SDK, descárguelo desde el [repositorio de GitHub](#) con `go get`:

```
go get github.com/aws/aws-xray-sdk-go
```

Para instrumentar clientes SDK de AWS, pase el cliente al método `xray.AWS()`. Luego puede hacer un seguimiento de las llamadas mediante la versión `WithContext` del método.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Una vez agregadas las dependencias correctas y realizados los cambios de código necesarios, active el seguimiento en la configuración de la función mediante la consola de Lambda o la API.

## Activación del seguimiento con la consola de Lambda

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

## Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

## Activación del seguimiento con la API de Lambda

Configure el rastreo en la función Lambda con AWS CLI o SDK de AWS, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Activación del seguimiento con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

Resources:



```
function:
  Type: AWS::Lambda::Function
  Properties:
    TracingConfig:
      Mode: Active
    ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
```

## Interpretación de un seguimiento de X-Ray

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.

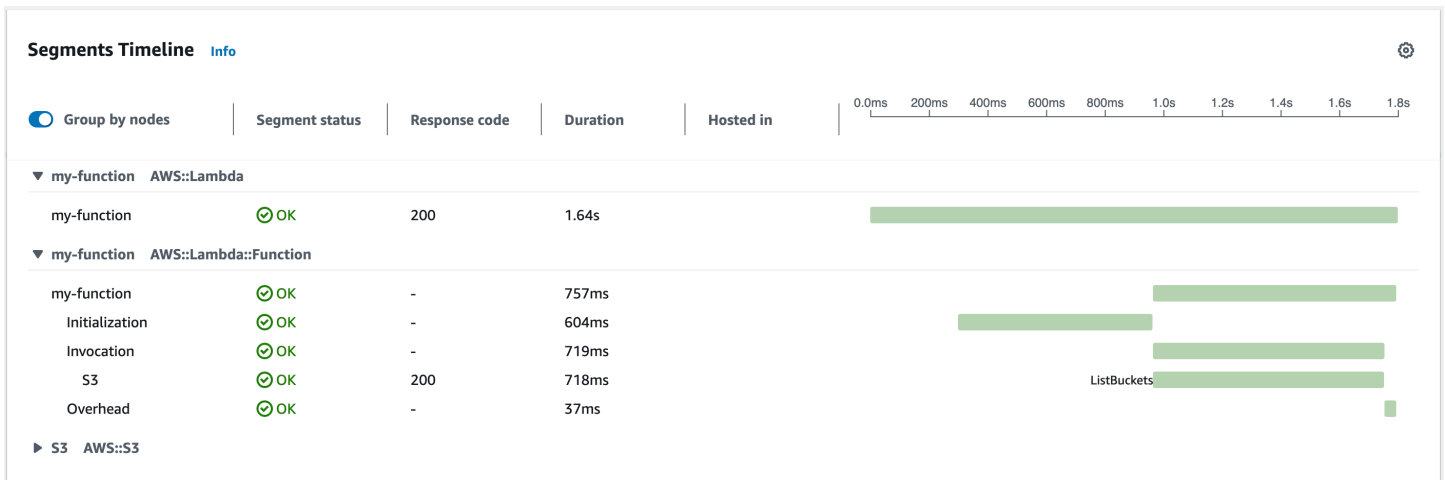


X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo, se muestra un seguimiento con estos dos segmentos. Ambos se denominan my-function, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar los tres subsegmentos.

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El rastro de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.
- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte el [SDK de AWS X-Ray para Go](#) en la Guía para desarrolladores de AWS X-Ray.

### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza cargos por almacenamiento y recuperación del seguimiento. Para obtener más información, consulte [Precios de AWS X-Ray](#).

## Creación de funciones Lambda con C#

Puede ejecutar la aplicación .NET en Lambda mediante los tiempos de ejecución administrados de .NET 6 o .NET 8, un tiempo de ejecución personalizado o una imagen de contenedor. Una vez compilado el código de la aplicación, puede implementarlo en Lambda como un archivo.zip o una imagen de contenedor. Lambda proporciona los siguientes tiempos de ejecución para lenguajes .NET:

### .NET

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
.NET 8	dotnet8	Amazon Linux 2023			
.NET 6	dotnet6	Amazon Linux 2	20 de diciembre de 2024	28 de febrero de 2025	31 de marzo de 2025

## Configuración del entorno de desarrollo de .NET

Para desarrollar y construir sus funciones de Lambda, puede usar cualquiera de los entornos de desarrollo integrado (IDE) de .NET, incluidos Microsoft Visual Studio, Visual Studio Code, y JetBrains Rider. Para simplificar su experiencia de desarrollo, AWS proporciona un conjunto de plantillas de proyecto .NET, así como la interfaz de línea de comandos (CLI) Amazon.Lambda.Tools.

Ejecute los siguientes comandos de la CLI de .NET para instalar estas plantillas de proyecto y las herramientas de línea de comandos.

### Instalación de las plantillas del proyecto .NET

Para instalar las plantillas del proyecto (.NET 8):

```
dotnet new install Amazon.Lambda.Templates
```

Para instalar las plantillas del proyecto (.NET 6):

```
dotnet new --install Amazon.Lambda.Templates
```

### Note

Si utiliza el tiempo de ejecución de Lambda administrado por .NET 6, le recomendamos que lo actualice para utilizar .NET 8. Para obtener más información, consulte [Administrar las actualizaciones de tiempo de ejecución de AWS Lambda](#) e [Introducción al tiempo de ejecución de .NET 8 para AWS Lambda](#) en el blog de informática de AWS.

## Instalación y actualización de las herramientas de la CLI

Ejecute los siguientes comandos para instalar, actualizar y desinstalar la CLI de `Amazon.Lambda.Tools`.

Para instalar las herramientas de línea de comandos:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para actualizar las herramientas de línea de comandos:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Para desinstalar las herramientas de línea de comandos:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

# Definir el controlador de funciones de Lambda en C#

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

Cuando se invoca la función y Lambda ejecuta el método controlador de la función, pasa dos argumentos a la función. El primer argumento es el objeto de event. Cuando otro Servicio de AWS invoca la función, el objeto event contiene datos sobre el evento que provocó la invocación de la función. Por ejemplo, un objeto event de API Gateway contiene información sobre la ruta, el método HTTP y los encabezados HTTP. La estructura exacta del evento varía en función del Servicio de AWS que esté invocando la función. Consulte [Integración de otros servicios](#) para obtener más información sobre los formatos de eventos para los servicios individuales.

Lambda también pasa un objeto context a la función. El objeto proporciona información sobre la invocación, la función y el entorno de ejecución. Para obtener más información, consulte [the section called "Context"](#).

El formato nativo de todos los eventos de Lambda son secuencias de bytes que representan el evento con formato JSON. A menos que los parámetros de entrada y salida de la función sean del tipo `System.IO.Stream`, debe serializarlos. Especifique el serializador que quiere usar configurando el atributo de conjunto `LambdaSerializer`. Para obtener más información, consulte [the section called "Serialización de las funciones de Lambda"](#).

## Temas

- [Modelos de ejecución de .NET para Lambda](#)
- [Controladores de bibliotecas de clases](#)
- [Controladores de conjuntos ejecutables](#)
- [Serialización de las funciones de Lambda](#)
- [Simplifique el código de funciones con el marco de anotaciones Lambda](#)
- [Restricciones del controlador de funciones de Lambda](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en C#](#)

## Modelos de ejecución de .NET para Lambda

Existen dos modelos de ejecución diferentes para ejecutar funciones de Lambda en .NET: el enfoque de biblioteca de clases y el enfoque de conjunto ejecutable.

En el enfoque de biblioteca de clases, se proporciona a Lambda una cadena que indica el `AssemblyName`, `ClassName` y `Method` de la función que se va a invocar. Para obtener más información sobre el formato de esta cadena, consulte [the section called “Controladores de bibliotecas de clases”](#). Durante la fase de inicialización de la función, se inicializa la clase de la función y se ejecuta cualquier código del constructor.

En el enfoque de conjunto ejecutable, se utiliza la característica de [declaraciones de nivel superior](#) de C# 9. Este enfoque genera un conjunto ejecutable que Lambda ejecuta cada vez que recibe un comando `invoke` para su función. Solo debe proporcionar a Lambda el nombre del conjunto ejecutable que se va a ejecutar.

En las siguientes secciones se ofrece un ejemplo de código de función para estos dos enfoques.

## Controladores de bibliotecas de clases

El siguiente código de función de Lambda muestra un ejemplo de un método de controlador (`FunctionHandler`) para una función de Lambda que utiliza el enfoque de biblioteca de clases. En esta función de ejemplo, Lambda recibe un evento de API Gateway que invoca la función. La función lee un registro de una base de datos y lo devuelve como parte de la respuesta de API Gateway.

```
[assembly:
  LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);
```



```
return new APIGatewayProxyResponse
{
    StatusCode = (int)HttpStatusCode.OK,
    Body = JsonSerializer.Serialize(databaseRecord)
};
}
```

Al crear una función de Lambda, debe proporcionar a Lambda información sobre el controlador de la función en forma de cadena de controlador. Esto le indica a Lambda qué método del código debe ejecutar cuando se invoque la función. En C#, el formato de la cadena del controlador cuando se utiliza el enfoque de biblioteca de clases es el siguiente:

ASSEMBLY::TYPE::METHOD, donde:

- ASSEMBLY es el nombre del archivo de conjunto de .NET para la aplicación. Si utiliza la CLI de `Amazon.Lambda.Tools` para compilar la aplicación y no establece el nombre del ensamblado con la propiedad `AssemblyName` del archivo `.csproj`, ASSEMBLY será simplemente el nombre de la carpeta que contiene el archivo `.csproj`.
- TYPE es el nombre completo del tipo de controlador, que consta de `Namespace` y `ClassName`.
- METHOD es el nombre del método de controlador en el código de la función.

En el código de ejemplo que se muestra, si se nombra el conjunto como `GetProductHandler`, entonces la cadena del controlador sería `GetProductHandler::GetProductHandler.Function::FunctionHandler`.

## Controladores de conjuntos ejecutables

En el siguiente ejemplo, la función de Lambda se define como un conjunto ejecutable. El método controlador de este código recibe el nombre `Handler`. Cuando se utilizan ensamblados ejecutables, se debe iniciar el tiempo de ejecución de Lambda. Para ello, se utiliza el método `LambdaBootstrapBuilder.Create`. Este método toma como entradas el método que su función usa como controlador y el serializador Lambda que debe usar.

Para obtener más información sobre el uso de instrucciones de nivel superior, consulte [Introducción del tiempo de ejecución .NET 6 de AWS Lambda](#) en el Blog de computación de AWS.

```
namespace GetProductHandler;
```

```
IDatabaseRepository repo = new DatabaseRepository();

await LambdaBootstrapBuilder.Create<APIGatewayProxyRequest>(Handler, new
    DefaultLambdaJsonSerializer())
    .Build()
    .RunAsync();

async Task<APIGatewayProxyResponse> Handler(APIGatewayProxyRequest apigProxyEvent,
    ILambdaContext context)
{
    var id = input.PathParameters["id"];

    var databaseRecord = await this.repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
};
```

Cuando se utilizan conjuntos ejecutables, la cadena del controlador que indica a Lambda cómo ejecutar el código es el nombre del conjunto. En este ejemplo, eso sería `GetProductHandler`.

## Serialización de las funciones de Lambda

Si las funciones de Lambda que utilizan tipos de entrada o salida distintos de un objeto `Stream`, debe agregar una biblioteca de serialización a la aplicación. Puede implementar la serialización mediante la serialización estándar basada en la reflexión proporcionada por `System.Text.Json` y `Newtonsoft.Json`, o mediante la serialización [generada en la fuente](#).

### Uso de la serialización generada en la fuente

La serialización generada en la fuente es una característica de las versiones 6 y posteriores de `.NET` que permite generar el código de serialización en tiempo de compilación. Elimina la necesidad de reflexión y puede mejorar el rendimiento de la función. Para utilizar la serialización generada en la fuente en la función, haga lo siguiente:

- Cree una nueva clase parcial que herede de `JsonSerializerContext` al añadir atributos `JsonSerializable` para todos los tipos que requieran serialización o deserialización.

- Configurar la `LambdaSerializer` para usar un `SourceGeneratorLambdaJsonSerializer<T>`.
- Actualice cualquier serialización o deserialización manual del código de su aplicación para usar la clase recién creada.

En el siguiente código se muestra un ejemplo de función que utiliza la serialización generada en la fuente.

```
[assembly:
  LambdaSerializer(typeof(SourceGeneratorLambdaJsonSerializer<CustomSerializer>))]

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord,
CustomSerializer.Default.Product)
        };
    }
}

[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{
```

```
}
```

### Note

Si desea utilizar la compilación anticipada (AOT) nativa con Lambda, debe usar la serialización generada en la fuente.

## Uso de la serialización basada en la reflexión

AWS proporciona bibliotecas prediseñadas que le permiten añadir rápidamente la serialización a su aplicación. Esto se configura mediante los paquetes NuGet `Amazon.Lambda.Serialization.SystemTextJson` o `Amazon.Lambda.Serialization.Json`. Entre bastidores, `Amazon.Lambda.Serialization.SystemTextJson` utiliza `System.Text.Json` para realizar tareas de serialización y `Amazon.Lambda.Serialization.Json` utiliza el paquete `Newtonsoft.Json`.

También puede crear su propia biblioteca de serialización mediante la implementación de la interfaz `ILambdaSerializer`, que está disponible como parte de la biblioteca `Amazon.Lambda.Core`. Esta interfaz define dos métodos:

- `T Deserialize<T>(Stream requestStream);`

Puede implementar este método para deserializar la carga de solicitud desde la API `Invoke` en el objeto que se pasa al controlador de la función de Lambda.

- `T Serialize<T>(T response, Stream responseStream);`

Puede implementar este método para serializar el resultado que devuelve el controlador de la función de Lambda en la carga de respuesta que devuelve la API de operación `Invoke`.

## Simplifique el código de funciones con el marco de anotaciones Lambda

Las anotaciones Lambda son un marco de trabajo para .NET 6 y .NET 8 que simplifica la escritura de funciones de Lambda mediante C#. Con el marco de anotaciones, puede reemplazar gran parte del código de una función de Lambda escrita con el modelo de programación normal. El código escrito con el marco utiliza expresiones más sencillas que le permiten centrarse en la lógica empresarial.

El siguiente código de ejemplo muestra cómo el uso del marco de anotaciones puede simplificar la escritura de funciones de Lambda. El primer ejemplo muestra el código escrito con el modelo de programa Lambda normal y el segundo muestra el equivalente con el marco de anotaciones.

```
public APIGatewayHttpApiV2ProxyResponse LambdaMathAdd(APIGatewayHttpApiV2ProxyRequest
    request, ILambdaContext context)
{
    if (!request.PathParameters.TryGetValue("x", out var xs))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    if (!request.PathParameters.TryGetValue("y", out var ys))
    {
        return new APIGatewayHttpApiV2ProxyResponse
        {
            StatusCode = (int)HttpStatusCode.BadRequest
        };
    }
    var x = int.Parse(xs);
    var y = int.Parse(ys);
    return new APIGatewayHttpApiV2ProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = (x + y).ToString(),
        Headers = new Dictionary<string, string> { { "Content-Type", "text/plain" } }
    };
}
```

```
[LambdaFunction]
[HttpApi(LambdaHttpMethod.Get, "/add/{x}/{y}")]
public int Add(int x, int y)
{
    return x + y;
}
```

Para ver otro ejemplo de cómo el uso de anotaciones Lambda puede simplificar su código, consulte este [ejemplo de aplicación multiservicio](#) en el repositorio de GitHub `awsdocs/aws-doc-sdk-examples`. La carpeta `PamApiAnnotations` usa anotaciones Lambda en el archivo principal

`function.cs`. A modo de comparación, la carpeta `PamApi` tiene archivos equivalentes escritos con el modelo de programación normal de Lambda.

El marco de anotaciones utiliza [generadores de código fuente](#) para generar código que se traduce del modelo de programación Lambda al código que se ve en el segundo ejemplo.

Para obtener más información acerca de cómo utilizar anotaciones de Lambda para .NET, consulte los recursos siguientes:

- El [aws/aws-lambda-dotnet](#) del repositorio de GitHub.
- [Presentamos el marco de anotaciones de Lambda para .NET \(versión preliminar\)](#) en el blog de herramientas para desarrolladores AWS.
- El paquete NuGet [Amazon.Lambda.Annotations](#).

## Inyección de dependencias con el marco de anotaciones Lambda

También puede utilizar el marco de anotaciones de Lambda para añadir una inyección de dependencias a las funciones de Lambda mediante una sintaxis con la que esté familiarizado. Al añadir un atributo `[LambdaStartup]` a un archivo `Startup.cs`, la estructura de anotaciones Lambda generará el código necesario en tiempo de compilación.

```
[LambdaStartup]
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
    }
}
```

La función de Lambda puede inyectar servicios mediante la inyección de un constructor o mediante la inyección en métodos individuales mediante el atributo `[FromServices]`.

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer)]

namespace GetProductHandler;
```

```
public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(IDatabaseRepository repo)
    {
        this._repo = repo;
    }

    [LambdaFunction]
    [HttpApi(LambdaHttpMethod.Get, "/product/{id}")]
    public async Task<Product> FunctionHandler([FromServices] IDatabaseRepository repository, string id)
    {
        return await this._repo.GetById(id);
    }
}
```

## Restricciones del controlador de funciones de Lambda

Tenga en cuenta que existen algunas restricciones que afectan a la firma del controlador.

- No puede ser `unsafe` ni utilizar tipos de puntero en la signatura del controlador, aunque puede utilizar el contexto `unsafe` dentro del método del controlador y sus dependencias. Para obtener más información, consulte [inseguro \(Referencia de C#\)](#) en el sitio web de Microsoft Docs.
- No puede pasar un número variable de parámetros utilizando la palabra clave `params`, ni utilizar `ArgIterator` como parámetro de entrada o de retorno que se utiliza para admitir un número variable de parámetros.
- El controlador no puede ser un método genérico, por ejemplo, `IList<T> Sort<T>(IList<T> input)`.
- No se admiten los controladores asíncronos con la signatura `async void`.

## Prácticas recomendadas de codificación para las funciones de Lambda en C#

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad.

- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución AWS Lambda contiene varias bibliotecas. Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, empaquete todas las dependencias con el paquete de implementación.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación. En las funciones creadas en .NET, evite cargar toda la biblioteca del SDK de AWS como parte del paquete de implementación. En lugar de ello, cree dependencias selectivas de los módulos que seleccionen los componentes del SDK que necesita (por ejemplo, DynamoDB, módulos del SDK de Amazon S3 y bibliotecas básicas de Lambda).
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.



- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Crear e implementar funciones de Lambda C# con archivos de archivo .zip

Un paquete de implementación .NET (archivo de archivo .zip) contiene el ensamblado compilado de su función junto con todas sus dependencias de ensamblado. El paquete también contiene un archivo `proj.deps.json`. Esto indica al tiempo de ejecución .NET todas las dependencias de su función y un archivo `proj.runtimeconfig.json`, que se utiliza para configurar el tiempo de ejecución.

Para implementar funciones individuales de Lambda, puede usar la `Amazon.Lambda.Tools` CLI de .NET Lambda Global. El uso del comando `dotnet lambda deploy-function` crea automáticamente un paquete de implementación .zip y lo implementa en Lambda. Sin embargo, le recomendamos que utilice marcos como AWS Serverless Application Model (AWS SAM) o el AWS Cloud Development Kit (AWS CDK) para implementar sus aplicaciones.NET en AWS.

Las aplicaciones sin servidor suelen incluir una combinación de funciones de Lambda y otros servicios administrados de Servicios de AWS que trabajan juntos para realizar una tarea empresarial determinada. AWS SAM y AWS CDK simplifican la creación e implementación de funciones de Lambda con otros Servicios de AWS a escala. La [especificación de plantillas de AWS SAM](#) proporciona una sintaxis sencilla y organizada para describir las funciones de Lambda, las API, los permisos, las configuraciones y los otros recursos de AWS que constituyen la aplicación sin servidor. Con [AWS CDK](#) puede definir la infraestructura de nube como código para ayudarle a crear aplicaciones fiables, escalables y rentables en la nube mediante marcos y lenguajes de programación modernos, como .NET. Tanto AWS CDK como AWS SAM utilizan la CLI de .NET Lambda Global para empaquetar sus funciones.

Si bien es posible utilizar [Capas de Lambda](#) con funciones en C# al [utilizar la CLI de .NET Core](#), se recomienda que no lo haga. Las funciones en C# que utilizan capas cargan de forma manual los ensamblajes compartidos en la memoria durante el [Fase "init"](#), lo que puede aumentar los tiempos de arranque en frío. En su lugar, incluya todo el código compartido en el momento de la compilación para aprovechar las optimizaciones integradas en el compilador de .NET.

En las siguientes secciones, encontrará instrucciones para crear e implementar funciones de Lambda .NET mediante AWS SAM, AWS CDK y la CLI de .NET Lambda Global y la CLI.

## Temas

- [Uso de la CLI de .NET Lambda Global](#)

- [Implemente las funciones de Lambda C# utilizando AWS SAM](#)
- [Implemente las funciones de Lambda C# utilizando AWS CDK](#)
- [Implementación de aplicaciones ASP .NET](#)

## Uso de la CLI de .NET Lambda Global

La CLI de .NET y la extensión .NET Lambda Global Tools (Amazon.Lambda.Tools) ofrecen una forma multiplataforma para crear aplicaciones Lambda basadas en .NET, empaquetarlas e implementarlas en Lambda. En esta sección, aprenderá a crear nuevos proyectos .NET de Lambda mediante la CLI de .NET y las plantillas de Amazon Lambda, y a empaquetarlos e implementarlos mediante Amazon.Lambda.Tools

### Temas

- [Requisitos previos](#)
- [Creación de proyectos .NET mediante la CLI de .NET](#)
- [Implementación de proyectos.NET mediante la CLI de .NET](#)
- [Uso de capas de Lambda con la CLI de .NET](#)

## Requisitos previos

### SDK para .NET 8

Si aún no lo ha hecho, instale el SDK y el tiempo de ejecución de [.NET 8](#).

### AWS Plantillas de proyecto .NET Amazon.Lambda.Templates

Para generar el código de la función de Lambda, utilice el paquete NuGet [Amazon.Lambda.Templates](#). Para instalar este paquete de plantillas, ejecute el siguiente comando:

```
dotnet new install Amazon.Lambda.Templates
```

### AWS Herramientas globales para la CLI de .NET Amazon.Lambda.Tools

Para crear sus funciones de Lambda, utilice la [extensión de herramientas globales de .NET Amazon.Lambda.Tools](#). Ejecute el siguiente comando para instalar Amazon.Lambda.Tools:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obtener más información sobre la Amazon.Lambda.Tools de la extensión de la CLI de .NET, consulte el repositorio en GitHub [Extensiones de AWS para la CLI de .NET](#).

## Creación de proyectos .NET mediante la CLI de .NET

En la CLI de .NET se usa el comando `dotnet new` para crear proyectos .NET desde la línea de comando. Lambda ofrece plantillas adicionales mediante el paquete [Amazon.Lambda.Templates](#) NuGet.

Tras la instalación de este paquete, ejecute el siguiente comando para ver una lista de las plantillas disponibles.

```
dotnet new list
```

Para examinar los detalles acerca de una plantilla, utilice la opción `help`. Por ejemplo, para ver los detalles de la plantilla `lambda.EmptyFunction`, ejecute el siguiente comando.

```
dotnet new lambda.EmptyFunction --help
```

Para crear una plantilla básica para una función de Lambda .NET, utilice la plantilla `lambda.EmptyFunction`. Esto crea una función sencilla que toma una cadena como entrada y la convierte a mayúsculas mediante el método `ToUpper`. Esta plantilla es compatible con las siguientes opciones:

- `--name`: el nombre de la función.
- `--region`: la región AWS para crear la función dentro.
- `--profile`: el nombre de un perfil en su archivo de credenciales de AWS SDK for .NET.

Para obtener más información sobre los perfiles de credenciales en .NET, consulte [Configurar credenciales AWS](#) en SDK AWS para Guía para desarrolladores de .NET.

En este ejemplo, creamos una nueva función vacía llamada `myDotnetFunction` con el perfil y la configuración Región de AWS predeterminados:

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Este comando crea los siguientes archivos y directorios en el directorio de su proyecto.

```
### myDotnetFunction
  ### src
  #   ### myDotnetFunction
  #     ### Function.cs
  #     ### Readme.md
  #     ### aws-lambda-tools-defaults.json
  #     ### myDotnetFunction.csproj
  ### test
    ### myDotnetFunction.Tests
    ### FunctionTest.cs
    ### myDotnetFunction.Tests.csproj
```

En el directorio `src/myDotnetFunction`, examine los archivos siguientes:

- `aws-lambda-tools-defaults.json`: aquí se especifican las opciones de línea de comando al implementar su función de Lambda. Por ejemplo:

```
"profile" : "default",
"region" : "us-east-2",
"configuration" : "Release",
"function-architecture": "x86_64",
"function-runtime":"dotnet8",
"function-memory-size" : 256,
"function-timeout" : 30,
"function-handler" : "myDotnetFunction::myDotnetFunction.Function::FunctionHandler"
```

- `Function.cs`: el código de la función de controlador de Lambda. Es una plantilla de C# que incluye la biblioteca `Amazon.Lambda.Core` predeterminada y un atributo `LambdaSerializer` predeterminado. Para obtener más información acerca de las opciones y los requisitos de serialización, consulte [Serialización de las funciones de Lambda](#). También incluye una función de muestra que puede editar para aplicar el código de su función de Lambda.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted into
// a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace myDotnetFunction;
```

```

public class Function
{
    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary#
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

```

- myDotnetFunction.csproj: un archivo [MSBuild](#) que enumera los archivos y ensamblados que componen la aplicación.

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
    <!-- This property makes the build directory similar to a publish directory and
    helps the AWS .NET Lambda Mock Test Tool find project dependencies. -->
    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
    <!-- Generate ready to run images during publishing to improve cold start time.
    -->
    <PublishReadyToRun>true</PublishReadyToRun>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
    Version="2.4.0" />
  </ItemGroup>
</Project>

```

- Readme: use este archivo para documentar su función de Lambda.

En el directorio myfunction/test, examine los archivos siguientes:

- `myDotnetFunction.Tests.csproj`: como se ha indicado con anterioridad, este es un archivo [MSBuild](#) que enumera los archivos y los ensamblados que componen el proyecto de prueba. Tenga en cuenta que también incluye la biblioteca `Amazon.Lambda.Core`, de modo que puede integrar a la perfección cualquier plantilla de Lambda necesaria para probar la función.

```
<Project Sdk="Microsoft.NET.Sdk">
  ...

  <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
  ...
```

- `FunctionTest.cs`: el mismo archivo de plantilla de código C# que se incluye en el directorio `src`. Edite este archivo para reflejar el código de producto de su función y probarlo antes de cargar su función de Lambda en un entorno de producción.

```
using Xunit;
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {
            // Invoke the lambda function and confirm the string was upper cased.
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

## Implementación de proyectos.NET mediante la CLI de .NET

Para crear su paquete de implementación e implementarlo en Lambda, utilice las herramientas de CLI Amazon.Lambda.Tools. Para implementar la función a partir de los archivos que creó en los pasos anteriores, primero navegue hasta la carpeta que contiene el archivo .csproj de la función.

```
cd myDotnetFunction/src/myDotnetFunction
```

Para implementar el código en Lambda como un paquete de implementación .zip, ejecute el siguiente comando. Elija su propio nombre de función.

```
dotnet lambda deploy-function myDotnetFunction
```

Durante la implementación, el asistente le pide que seleccione un [the section called “Rol de ejecución \(permisos para que las funciones accedan a otros recursos\)”](#). Para este ejemplo, seleccione `lambda_basic_role`.

Una vez implementada la función, puede probarla en la nube con el comando `dotnet lambda invoke-function`. Para el código de ejemplo de la plantilla `lambda.EmptyFunction`, puede probar su función pasando una cadena mediante la opción `--payload`.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything is OK"
```

Si la función se ha implementado correctamente, debería ver un resultado similar al siguiente.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything is OK"
Amazon Lambda Tools for .NET Core applications (5.8.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/aws/aws-lambda-dotnet

Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id Duration: 0.99 ms          Billed Duration: 1 ms          Memory
Size: 256 MB          Max Memory Used: 12 MB
```



## Uso de capas de Lambda con la CLI de .NET

### Note

El uso de capas con funciones en un lenguaje compilado, como C#, puede no ofrecer las mismas ventajas que con un lenguaje interpretado, como Python. Como C# es un lenguaje compilado, las funciones aún tienen que cargar de forma manual los ensamblajes compartidos en la memoria durante la fase de inicio, lo que puede aumentar los tiempos de arranque en frío. En su lugar, se recomienda incluir cualquier código compartido en el momento de la compilación para aprovechar las optimizaciones del compilador integradas.

La CLI de .NET admite comandos que lo ayudan a publicar capas e implementar funciones de C# que consumen capas. Para publicar una capa en un bucket de Amazon S3 específico, ejecute el siguiente comando en el mismo directorio que su archivo `.csproj`:

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-bucket <s3_bucket_name>
```

Luego, cuando implemente la función mediante la CLI de .NET, especifique el ARN de la capa que consumirá en el siguiente comando:

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-east-1:123456789012:layer:layer-name:1
```

Para conocer un ejemplo completo de una función “Hola, mundo”, consulte la muestra [blank-csharp-with-layer](#).

## Implemente las funciones de Lambda C# utilizando AWS SAM

AWS Serverless Application Model (AWS SAM) es un conjunto de herramientas que ayuda a agilizar el proceso de creación y ejecución de aplicaciones sin servidor en AWS. Defina los recursos de su aplicación en una plantilla YAML o JSON y utilice la interfaz de la línea de comandos de AWS SAM (AWS SAM CLI) para crear, empaquetar e implementar sus aplicaciones. Al crear una función de Lambda a partir de una plantilla de AWS SAM, AWS SAM crea automáticamente un paquete de implementación `.zip` o una imagen de contenedor con el código de la función y las dependencias que especifique. Después, AWS SAM implementa la función mediante una [pila AWS CloudFormation](#).zip

Para obtener más información sobre el uso de AWS SAM para crear e implementar funciones de Lambda, consulte [Introducción a AWS SAM](#) en la Guía para desarrolladores de AWS Serverless Application Model.

Los siguientes pasos muestran cómo descargar, crear e implementar una aplicación Hello World .NET de muestra mediante AWS SAM. Esta aplicación de muestra utiliza una función de Lambda y un punto de conexión de Amazon API Gateway para implementar un backend de API básico. Cuando envía una solicitud GET de HTTP a su punto de conexión de API Gateway, la última invoca la función de Lambda. La función devuelve un mensaje “Hello world”, junto con la dirección IP de la instancia de la función de Lambda que procesa la solicitud.

Al crear e implementar la aplicación mediante AWS SAM, entre bastidores, la CLI AWS SAM utiliza el comando `dotnet lambda package` para empaquetar los grupos de códigos de las funciones de Lambda individuales.

## Requisitos previos

SDK para .NET 8

Instale el SDK y el tiempo de ejecución de [.NET 8](#).

Versión 1.39 o posterior de la CLI de AWS SAM

Para instalar la última versión de la CLI de AWS SAM, consulte [Instalación de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación con la plantilla Hello world de .NET con el siguiente comando.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Este comando crea los siguientes archivos y directorios en el directorio de su proyecto.

```
### sam-app
### README.md
### events
#   ### event.json
### omnisharp.json
```

```
### samconfig.toml
### src
#   ### HelloWorld
#       ### Function.cs
#       ### HelloWorld.csproj
#       ### aws-lambda-tools-defaults.json
### template.yaml
### test
    ### HelloWorld.Test
        ### FunctionTest.cs
        ### HelloWorld.Tests.csproj
```

2. Navegue hasta el directorio que contiene el `template.yaml` file. Este archivo es una plantilla que define los recursos AWS de la aplicación, incluida la función de Lambda y una API de API Gateway.

```
cd sam-app
```

3. Para crear la fuente de la aplicación, ejecute el siguiente comando.

```
sam build
```

4. Para implementar la aplicación en AWS, ejecute el siguiente comando.

```
sam deploy --guided
```

Este comando empaqueta e implementa la aplicación con la siguiente serie de solicitudes. Para aceptar las opciones predeterminadas, pulse Enter.

#### Note

Puede que HelloWorldFunction no tenga definida la autorización, ¿está bien? Asegúrese de ingresar y.

- Nombre de la pila: el nombre de la pila para la implementación en AWS CloudFormation. Este nombre debe ser exclusivo para Cuenta de AWS y Región de AWS.
- Región de AWS: El Región de AWS en el que desea implementar su aplicación.
- Confirme los cambios antes de la implementación: seleccione “Sí” para revisar manualmente cualquier conjunto de cambios antes de que AWS SAM implemente los cambios en la

aplicación. Si selecciona “No”, la CLI AWS SAM implementa automáticamente los cambios en las aplicaciones.

- Permitir la creación de roles de IAM en la CLI de SAM: muchas plantillas AWS SAM, incluida la de Hello world en este ejemplo, crean roles (de IAM) AWS Identity and Access Management para dar permiso a las funciones de Lambda para acceder a otros Servicios de AWS. Seleccione “Sí” para conceder permiso para implementar una pila AWS CloudFormation que cree o modifique los roles de IAM.
  - Desactivar la reversión: de forma predeterminada, si AWS SAM encuentra un error durante la creación o la implementación de la pila, la devuelve a la versión anterior. Seleccione “No” para aceptar este valor predeterminado.
  - Puede que HelloWorldFunction no tenga definida la autorización, ¿está bien?: Ingrese y.
  - Guardar los argumentos en samconfig.toml: seleccione “Sí” para guardar las opciones de configuración. En el futuro, podrá volver a ejecutar `sam deploy` sin parámetros para implementar cambios en la aplicación.
5. Una vez finalizada la implementación de la aplicación, la CLI devuelve el nombre de recurso de Amazon (ARN) de la función de Lambda de Hello World y el rol de IAM creado para la misma. También muestra el punto de conexión de la API de API Gateway. Para probar la aplicación, abra el punto de conexión en un navegador. Verá una respuesta parecida a la siguiente.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Use el siguiente comando para eliminar los recursos. Tenga en cuenta que el punto de conexión de la API que ha creado es un punto de conexión público al que se puede acceder a través de Internet. Se recomienda eliminar este punto de conexión después de las pruebas.

```
sam delete
```

## Siguientes pasos

Para obtener más información sobre AWS SAM para crear e implementar funciones de Lambda mediante .NET, consulte los siguientes recursos:

- [La guía para desarrolladores de AWS Serverless Application Model \(AWS SAM\)](#)
- [Creación de aplicaciones .NET sin servidor con la CLI AWS Lambda de SAM](#)

# Implemente las funciones de Lambda C# utilizando AWS CDK

AWS Cloud Development Kit (AWS CDK) es un marco de desarrollo de software de código abierto para definir la infraestructura de nube como código con marcos y lenguajes de programación modernos, como .NET. Los proyectos AWS CDK se ejecutan para generar plantillas AWS CloudFormation que luego se utilizan para implementar el código.

Para crear e implementar un ejemplo de aplicación Hello world .NET mediante AWS CDK, siga las instrucciones de las siguientes secciones. La aplicación de ejemplo implementa un backend de API básico que consta de un punto de conexión API Gateway y una función de Lambda. Cuando se envía una solicitud HTTP GET al punto de conexión, API Gateway invoca la función de Lambda. La función devuelve un mensaje “Hello world”, junto con la dirección IP de la instancia Lambda que procesa la solicitud.

## Requisitos previos

### SDK para .NET 8

Instale el SDK y el tiempo de ejecución de [.NET 8](#).

### Versión 2 de la AWS CDK

Para obtener información sobre cómo instalar la última versión de AWS CDK consulte [Introducción a la AWS CDK](#) en la Guía para desarrolladores de AWS Cloud Development Kit (AWS CDK) versión 2..

## Implementar una aplicación de ejemplo de AWS CDK

1. Cree un directorio de proyectos para la aplicación de ejemplo y navegue hasta él.

```
mkdir hello-world
cd hello-world
```

2. Inicialice una nueva aplicación AWS CDK mediante la ejecución del siguiente comando.

```
cdk init app --language csharp
```

El comando crea los siguientes archivos y directorios en el directorio del proyecto

```
### README.md
```

```
### cdk.json
### src
### HelloWorld
#   ### GlobalSuppressions.cs
#   ### HelloWorld.csproj
#   ### HelloWorldStack.cs
#   ### Program.cs
### HelloWorld.sln
```

- Abra el directorio `src` y cree una nueva función de Lambda mediante la CLI de .NET. Esta es la función que implementará mediante AWS CDK. En este ejemplo, se crea una función `HelloWorld` denominada `HelloWorldLambda` mediante la plantilla `lambda.EmptyFunction`.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Después de este paso, la estructura dentro del directorio de proyectos debería tener el siguiente aspecto.

```
### README.md
### cdk.json
### src
### HelloWorld
#   ### GlobalSuppressions.cs
#   ### HelloWorld.csproj
#   ### HelloWorldStack.cs
#   ### Program.cs
### HelloWorld.sln
### HelloWorldLambda
### src
#   ### HelloWorldLambda
#       ### Function.cs
#       ### HelloWorldLambda.csproj
#       ### Readme.md
#       ### aws-lambda-tools-defaults.json
### test
### HelloWorldLambda.Tests
### FunctionTest.cs
### HelloWorldLambda.Tests.csproj
```

- Abra el archivo `HelloWorldStack.cs` del directorio `src/HelloWorld`. Reemplace el contenido del archivo por lo siguiente.

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.Logs;
using Constructs;

namespace CdkTest
{
    public class HelloWorldStack : Stack
    {
        internal HelloWorldStack(Construct scope, string id, IStackProps props =
null) : base(scope, id, props)
        {
            var buildOption = new BundlingOptions()
            {
                Image = Runtime.DOTNET_8.BundlingImage,
                User = "root",
                OutputType = BundlingOutput.ARCHIVED,
                Command = new string[]{
function.zip"
                    "/bin/sh",
                    "-c",
                    " dotnet tool install -g Amazon.Lambda.Tools"+
                    " && dotnet build"+
                    " && dotnet lambda package --output-package /asset-output/
                };

            var helloWorldLambdaFunction = new Function(this,
"HelloWorldFunction", new FunctionProps
            {
                Runtime = Runtime.DOTNET_8,
                MemorySize = 1024,
                LogRetention = RetentionDays.ONE_DAY,
                Handler =
"HelloWorldLambda::HelloWorldLambda.Function::FunctionHandler",
                Code = Code.FromAsset("./src/HelloWorldLambda/src/
HelloWorldLambda", new Amazon.CDK.AWS.S3.Assets.AssetOptions
                {
                    Bundling = buildOption
                }
            }));
        }
    }
}
```

```
}
```

Este es el código para compilar y empaquetar el código de la aplicación, así como la definición de la propia función de Lambda. El objeto `BundlingOptions` permite crear un archivo zip junto con un conjunto de comandos que se utilizan para generar el contenido del archivo zip. En este caso, el comando `dotnet lambda package` se usa para compilar y generar el archivo zip.

5. Para implementar la aplicación, ejecute el siguiente comando.

```
cdk deploy
```

6. Invoque la función de Lambda implementada mediante la CLI de .NET Lambda.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Al acabar las prueba, a menos que desee retener los recursos que creó, puede eliminarlos. Use el siguiente comando para eliminar los recursos.

```
cdk destroy
```

## Siguientes pasos

Para obtener más información sobre AWS CDK para crear e implementar funciones de Lambda mediante .NET, consulte los siguientes recursos:

- [Utilización del CDK AWS en C#](#)
- [Cree, empaquete y publique funciones de Lambda C# de .NET con el CDK AWS](#)

## Implementación de aplicaciones ASP .NET

Además de alojar funciones basadas en eventos, también puede utilizar .NET con Lambda para alojar aplicaciones ASP .NET ligeras. Puede crear e implementar aplicaciones ASP .NET mediante el paquete NuGet `Amazon.Lambda.AspNetCoreServer`. En esta sección, aprenderá a implementar una API web de ASP .NET en Lambda mediante las herramientas de CLI de Lambda .NET.

### Temas

- [Requisitos previos](#)
- [Implementación de una API web de ASP.NET en Lambda](#)



- [Implementación de API mínimas de ASP .NET en Lambda](#)

## Requisitos previos

### SDK para .NET 8

Instale el SDK de [.NET 8](#) y el tiempo de ejecución de ASP .NET Core.

### Amazon.Lambda.Tools

Para crear sus funciones de Lambda, utilice la [extensión de herramientas globales de .NET Amazon.Lambda.Tools](#). Ejecute el siguiente comando para instalar Amazon.Lambda.Tools:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obtener más información sobre la Amazon.Lambda.Tools de la extensión de la CLI de .NET, consulte el repositorio en GitHub [Extensiones de AWS para la CLI de .NET](#).

### Amazon.Lambda.Templates

Para generar el código de la función de Lambda, utilice el paquete NuGet [Amazon.Lambda.Templates](#). Para instalar este paquete de plantillas, ejecute el siguiente comando:

```
dotnet new --install Amazon.Lambda.Templates
```

## Implementación de una API web de ASP.NET en Lambda

Para implementar una API web mediante ASP .NET, puede usar las plantillas .NET Lambda para crear un nuevo proyecto de API web. Utilice el siguiente comando para inicializar un nuevo proyecto de API web de ASP .NET. En el comando de ejemplo, asignamos un nombre al proyecto `AspNetOnLambda`.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Este comando crea los siguientes archivos y directorios en el directorio de su proyecto.

```
.  
### AspNetOnLambda
```

```
### src
#   ### AspNetOnLambda
#       ### AspNetOnLambda.csproj
#       ### Controllers
#       #   ### ValuesController.cs
#       ### LambdaEntryPoint.cs
#       ### LocalEntryPoint.cs
#       ### Readme.md
#       ### Startup.cs
#       ### appsettings.Development.json
#       ### appsettings.json
#       ### aws-lambda-tools-defaults.json
#       ### serverless.template
### test
    ### AspNetOnLambda.Tests
        ### AspNetOnLambda.Tests.csproj
        ### SampleRequests
        #   ### ValuesController-Get.json
        ### ValuesControllerTests.cs
        ### appsettings.json
```

Cuando Lambda invoca la función, el punto de entrada que utiliza es el archivo `LambdaEntryPoint.cs`. El archivo creado por la plantilla Lambda de .NET contiene el siguiente código.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

El punto de entrada utilizado por Lambda debe heredar de una de las tres clases base del paquete `Amazon.Lambda.AspNetCoreServer`. Estas tres clases base son:

- `APIGatewayProxyFunction`
- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

La clase predeterminada que se utiliza al crear el archivo `LambdaEntryPoint.cs` con la plantilla .NET Lambda proporcionada es `APIGatewayProxyFunction`. La clase base que utilice en la función depende de la capa de API situada delante de la función de Lambda.

Cada una de las tres clases base contiene un método público denominado `FunctionHandlerAsync`. El nombre de este método formará parte de la [cadena de controladores](#) que Lambda utiliza para invocar la función. El método `FunctionHandlerAsync` transforma la carga útil del evento entrante en el formato de ASP .NET correcto y la respuesta de ASP .NET en una carga útil de respuesta de Lambda. Para el proyecto `AspNetOnLambda` de ejemplo que se muestra, la cadena del controlador sería la siguiente.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Para implementar la API en Lambda, ejecute los siguientes comandos para navegar al directorio que contiene el archivo de código fuente e implementar la función mediante AWS CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

#### Tip

Al implementar una API mediante el comando `dotnet lambda deploy-serverless`, AWS CloudFormation asigna un nombre a la función de Lambda según el nombre de pila que especifique durante la implementación. Para asignar un nombre personalizado a la función de Lambda, edite el archivo `serverless.template` para agregar una propiedad `FunctionName` al recurso `AWS::Serverless::Function`. Para obtener más información, consulte [Tipo de nombre](#) en la Guía del usuario de AWS CloudFormation.

## Implementación de API mínimas de ASP .NET en Lambda

Para implementar una API mínima de ASP .NET en Lambda, puede usar las plantillas Lambda .NET para crear un nuevo proyecto de API mínima. Use el siguiente comando para inicializar un nuevo proyecto de API mínima. En este ejemplo, usamos el nombre de proyecto `MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

El comando crea los siguientes archivos y directorios en el directorio de su proyecto.

```
### MinimalApiOnLambda
### src
### MinimalApiOnLambda
### Controllers
# ### CalculatorController.cs
### MinimalApiOnLambda.csproj
### Program.cs
### Readme.md
### appsettings.Development.json
### appsettings.json
### aws-lambda-tools-defaults.json
### serverless.template
```

El archivo `Program.cs` contiene la salida siguiente.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();

// Add AWS Lambda support. When application is run in Lambda Kestrel is swapped out as
// the web server with Amazon.Lambda.AspNetCoreServer. This
// package will act as the webserver translating request and responses between the
// Lambda event source and ASP.NET Core.
builder.Services.AddAWSLambdaHosting(LambdaEventSource.RestApi);

var app = builder.Build();

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
```

```
app.MapGet("/", () => "Welcome to running ASP.NET Core Minimal API on AWS Lambda");  
  
app.Run();
```

Para configurar su API mínima para que se ejecute en Lambda, es posible que necesite editar este código para que las solicitudes y respuestas entre Lambda y ASP .NET Core se traduzcan correctamente. De forma predeterminada, la función está configurada para una fuente de eventos de la API de REST. En el caso de una API HTTP o un equilibrador de carga de aplicación, sustituya (`LambdaEventSource.RestApi`) por una de las siguientes opciones:

- (`LambdaEventSource.HttpApi`)
- (`LambdaEventSource.ApplicationLoadBalancer`)

Para implementar su API mínima en Lambda, ejecute los siguientes comandos para navegar al directorio que contiene el archivo de código fuente e implementar la función mediante AWS CloudFormation.

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda  
dotnet lambda deploy-serverless
```

# Implementar funciones de Lambda .NET con imágenes de contenedor

Hay tres formas de crear una imagen de contenedor para una función de Lambda en .NET:

- [Uso de una imagen base de AWS para .NET](#)

Las [imágenes base de AWS](#) vienen precargadas con un tiempo de ejecución de lenguaje, un cliente de interfaz de tiempo de ejecución para administrar la interacción entre Lambda y el código de la función y un emulador de interfaz de tiempo de ejecución para realizar pruebas a nivel local.

- [Uso de una imagen base exclusiva del sistema operativo de AWS](#)

[Las imágenes base exclusivas del sistema operativo de AWS](#) contienen una distribución de Amazon Linux y el [emulador de interfaz de tiempo de ejecución](#). Por lo general, estas imágenes se utilizan para crear imágenes contenedoras para lenguajes compilados, como [Go](#) y [Rust](#), y para un lenguaje o versión de un lenguaje para los que Lambda no proporciona una imagen base, como Node.js 19. También puede usar imágenes base exclusivas del sistema operativo para implementar un [tiempo de ejecución personalizado](#). Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para .NET](#) en la imagen.

- [Uso de una imagen base que no sea de AWS](#)

Puede utilizar una imagen base alternativa de otro registro de contenedores, como Alpine Linux o Debian. También puede utilizar una imagen personalizada creada por su organización. Para que la imagen sea compatible con Lambda, debe incluir el [cliente de interfaz de tiempo de ejecución para .NET](#) en la imagen.

## Tip

Para reducir el tiempo que tardan las funciones de contenedor de Lambda en activarse, consulte [Uso de compilaciones de varias fases](#) en la documentación de Docker. Para compilar imágenes de contenedores eficientes, siga [Prácticas recomendadas para escribir Dockerfiles](#).

En esta página, se explica cómo compilar, probar e implementar imágenes de contenedor para Lambda.

## Temas

- [AWS imágenes base para .NET](#)
- [Uso de una imagen base de AWS para .NET](#)
- [Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución](#)

## AWS imágenes base para .NET

AWS proporciona las siguientes imágenes base para .NET:

Etiqueta	Tiempo de ejecución	Sistema operativo	Dockerfile	Obsolescencia
8	.NET 8	Amazon Linux 2023	<a href="#">Dockerfile para .NET 8 en GitHub</a>	
6	.NET 6	Amazon Linux 2	<a href="#">Dockerfile para .NET 6 en GitHub</a>	20 de diciembre de 2024

Repositorio de Amazon ECR: [gallery.ecr.aws/lambda/dotnet](https://gallery.ecr.aws/lambda/dotnet)

## Uso de una imagen base de AWS para .NET

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [SDK de .NET](#): en los siguientes pasos se utiliza la imagen base de .NET 8. Asegúrese de que su versión de .NET coincida con la versión de la [imagen base](#) que especifique en su Dockerfile.
- [Docker](#)

### Creación e implementación de una imagen con una imagen base

En los siguientes pasos, utilizará [Amazon.Lambda.Templates](#) y [Amazon.Lambda.Tools](#) para crear un proyecto .NET. A continuación, creará una imagen de Docker, cargará la imagen a Amazon ECR y la implementará en una función de Lambda.

1. Instale el paquete NuGet de [Amazon.Lambda.Templates](#).

```
dotnet new install Amazon.Lambda.Templates
```

2. Cree un proyecto de .NET con la plantilla `lambda.image.EmptyFunction`.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

3. Vaya al directorio `MyFunction/src/MyFunction`. Aquí es donde se almacenan los archivos del proyecto. Examine los siguientes archivos:

- `aws-lambda-tools-defaults.json`: en este archivo se especifican las opciones de la línea de comandos al implementar su función de Lambda.
  - `Function.cs`: el código de la función de controlador de Lambda. Es una plantilla de C# que incluye la biblioteca `Amazon.Lambda.Core` predeterminada y un atributo `LambdaSerializer` predeterminado. Para obtener más información acerca de las opciones y los requisitos de serialización, consulte [Serialización de las funciones de Lambda](#). Puede utilizar el código proporcionado para realizar pruebas o sustituirlo por su propio código.
  - `MyFunction.csproj`: un [archivo de proyecto](#) de .NET que enumera los archivos y ensamblados que componen la aplicación.
  - `Readme.md`: este archivo contiene más información sobre la función de Lambda de muestra.
4. Examine el `Dockerfile` en el directorio `src/MyFunction`. Puede usar el `Dockerfile` proporcionado para realizar pruebas o sustituirlo por el suyo propio. Si usa el suyo propio, asegúrese de que se cumpla lo siguiente:
    - Establezca la propiedad `FROM` en el [URI de la imagen base](#). Su versión de .NET debe coincidir con la versión de la imagen base.
    - Establezca el argumento `CMD` para el controlador de la función de Lambda. Esto debe coincidir con `image-command` en `aws-lambda-tools-defaults.json`.

### Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM public.ecr.aws/lambda/dotnet:8

# Copy function code to Lambda-defined environment variable
COPY publish/* ${LAMBDA_TASK_ROOT}
```



```
# Set the CMD to your handler (could also be done as a parameter override outside
of the Dockerfile)
CMD [ "MyFunction::MyFunction.Function::FunctionHandler" ]
```

5. Instale la [herramienta global .NET](#) de Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Si ya se ha instalado Amazon.Lambda.Tools, asegúrese de que posee la versión más reciente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

6. Cambie el directorio a *MyFunction/src/MyFunction* si aún no se encuentra allí.

```
cd src/MyFunction
```

7. Utilice Amazon.Lambda.Tools para crear la imagen de Docker, subirla a un nuevo repositorio de Amazon ECR e implementar la función de Lambda.

En `--function-role`, especifique el nombre del rol (no el nombre de recurso de Amazon (ARN)) del [rol de ejecución](#) de la función. Por ejemplo, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Para obtener más información acerca de la herramienta global de .NET de Amazon.Lambda.Tools, consulte el repositorio [Extensiones de AWS para la CLI de .NET](#) en GitHub.

8. Invoque la función.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Si todo es correcto, verá lo siguiente:

```
Payload:
"TESTING THE FUNCTION"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
```

```
REPORT RequestId: id Duration: 0.99 ms Billed Duration: 1 ms Memory  
Size: 256 MB Max Memory Used: 12 MB
```

## 9. Elimine la función de Lambda.

```
dotnet lambda delete-function MyFunction
```

## Uso de una imagen base alternativa con el cliente de interfaz de tiempo de ejecución

Si usa una [imagen base exclusiva del sistema operativo](#) o una imagen base alternativa, debe incluir el cliente de interfaz de tiempo de ejecución en su imagen. El cliente de interfaz de tiempo de ejecución extiende el [Uso de la API de tiempo de ejecución de Lambda para tiempos de ejecución personalizados](#), que administra la interacción entre Lambda y el código de la función.

En el siguiente ejemplo, se muestra cómo crear una imagen de contenedor para .NET con una imagen base que no es de AWS y cómo agregar el [paquete Amazon.Lambda.RuntimeSupport](#), que es el cliente de interfaz de tiempo de ejecución de Lambda para .NET. El Dockerfile de ejemplo utiliza la imagen base de .NET 8 para Microsoft.

### Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [SDK de .NET](#) : en los siguientes pasos, se utiliza una imagen base de .NET 8. Asegúrese de que su versión de .NET coincida con la versión de la [imagen base](#) que especifique en su Dockerfile.
- [Docker](#)

## Creación e implementación de una imagen con una imagen base alternativa

1. Instale el paquete NuGet de [Amazon.Lambda.Templates](#).

```
dotnet new install Amazon.Lambda.Templates
```

2. Cree un proyecto de .NET con la plantilla `lambda.CustomRuntimeFunction`. Esta plantilla incluye el paquete [Amazon.Lambda.RuntimeSupport](#).

```
dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
```

3. Vaya al directorio `MyFunction/src/MyFunction`. Aquí es donde se almacenan los archivos del proyecto. Examine los siguientes archivos:
  - `aws-lambda-tools-defaults.json`: en este archivo se especifican las opciones de la línea de comandos al implementar su función de Lambda.
  - `Function.cs`: el código contiene una clase con un método `Main` que inicializa la biblioteca `Amazon.Lambda.RuntimeSupport` como el arranque. El método `Main` es el punto de entrada para el proceso de la función. El método `Main` empaqueta el controlador de funciones en un contenedor con el que puede funcionar el arranque. Para obtener más información, consulte [Uso de Amazon.Lambda.RuntimeSupport como biblioteca de clases](#) en el repositorio de GitHub.
  - `MyFunction.csproj`: un [archivo de proyecto](#) de .NET que enumera los archivos y ensamblados que componen la aplicación.
  - `Readme.md`: este archivo contiene más información sobre la función de Lambda de muestra.
4. Abra el archivo `aws-lambda-tools-defaults.json` y agregue las siguientes líneas:

```
"package-type": "image",  
"docker-host-build-output-dir": "./bin/Release/Lambda-publish"
```

- `package-type`: define el paquete de despliegue como una imagen de contenedor.
- `docker-host-build-output-dir`: establece el directorio de salida para el proceso de compilación.

#### Example `aws-lambda-tools-defaults.json`

```
{  
  "Information": [  
    "This file provides default values for the deployment wizard inside Visual  
    Studio and the AWS Lambda commands added to the .NET Core CLI.",  
    "To learn more about the Lambda commands with the .NET Core CLI execute the  
    following command at the command line in the project root directory.",  
    "dotnet lambda help",  
    "All the command line options for the Lambda command can be specified in this  
    file."  
  ],  
  "profile": "",  
  "region": "us-east-1",  
  "configuration": "Release",  
  "function-runtime": "provided.al2023",  
}
```

```

"function-memory-size": 256,
"function-timeout": 30,
"function-handler": "bootstrap",
"msbuild-parameters": "--self-contained true",
"package-type": "image",
"docker-host-build-output-dir": "./bin/Release/lambda-publish"
}

```

5. Cree un Dockerfile en el directorio `MyFunction/src/MyFunction`. El siguiente Dockerfile de ejemplo usa una imagen base de .NET para Microsoft en lugar de una [imagen base de AWS](#).
  - Establezca la propiedad FROM como el identificador de la imagen base. Su versión de .NET debe coincidir con la versión de la imagen base.
  - Utilice el comando COPY para copiar la función en el directorio `/var/task`.
  - Configure ENTRYPOINT como el módulo que desea que el contenedor de Docker ejecute cuando se inicie. En este caso, el módulo es el arranque, que inicializa la biblioteca `Amazon.Lambda.RuntimeSupport`.

### Example Dockerfile

```

# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM mcr.microsoft.com/dotnet/runtime:8.0

# Set the image's internal work directory
WORKDIR /var/task

# Copy function code to Lambda-defined environment variable
COPY "bin/Release/net8.0/linux-x64" .

# Set the entrypoint to the bootstrap
ENTRYPOINT ["/usr/bin/dotnet", "exec", "/var/task/bootstrap.dll"]

```

6. Instale la [extensión de herramientas globales de .NET](#) de Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Si ya se ha instalado Amazon.Lambda.Tools, asegúrese de que posee la versión más reciente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Utilice Amazon.Lambda.Tools para crear la imagen de Docker, subirla a un nuevo repositorio de Amazon ECR e implementar la función de Lambda.

En `--function-role`, especifique el nombre del rol (no el nombre de recurso de Amazon (ARN)) del [rol de ejecución](#) de la función. Por ejemplo, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Para obtener más información acerca de la extensión CLI de la herramienta global de .NET de Amazon Lambda, consulte el repositorio [Extensiones de AWS para la CLI de .NET](#) en GitHub.

8. Invoque la función.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Si todo es correcto, verá lo siguiente:

```
Payload:
"TESTING THE FUNCTION"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id Duration: 0.99 ms          Billed Duration: 1 ms          Memory
Size: 256 MB      Max Memory Used: 12 MB
```

9. Elimine la función de Lambda.

```
dotnet lambda delete-function MyFunction
```

# Compilar el código de una función de Lambda .NET en un formato de tiempo de ejecución nativo

.NET 8 admite la compilación nativa anticipada (AOT). Con el enfoque nativo anticipado, puede compilar el código de la función de Lambda en un formato de tiempo de ejecución nativo, lo que elimina la necesidad de compilar código .NET en el tiempo de ejecución. La compilación nativa anticipada puede reducir el tiempo de inicio en frío de las funciones de Lambda que crea con .NET. Para obtener más información, consulte [Introducción al tiempo de ejecución de .NET 8 para AWS Lambda](#) en el Blog de informática de AWS.

## Secciones

- [Tiempo de ejecución de Lambda](#)
- [Requisitos previos](#)
- [Introducción](#)
- [Serialización](#)
- [Recorte](#)
- [Resolución de problemas](#)

## Tiempo de ejecución de Lambda

Utilice el tiempo de ejecución administrado de .NET 8 en Lambda para implementar una función de Lambda que se cree con la compilación nativa anticipada (AOT). Este tiempo de ejecución es compatible con las arquitecturas x86\_64 y arm64.

Cuando implementa una función de Lambda en .NET sin utilizar AOT, la aplicación primero se compila en código de lenguaje intermedio (IL). Durante el tiempo de ejecución, la compilación en tiempo de ejecución (JIT) de Lambda toma el código IL y lo compila en código de máquina según sea necesario. Con una función de Lambda que se compila con antelación mediante una AOT nativa, el código se compila en código de máquina al implementar la función. Por lo tanto, no se depende del tiempo de ejecución de .NET ni del SDK en el tiempo de ejecución de Lambda para compilar el código antes de la ejecución.

Una limitación de AOT implica que el código de la aplicación debe compilarse en un entorno con el mismo sistema operativo Amazon Linux 2023 (AL2023) que utiliza el tiempo de ejecución de .NET 8. La CLI de Lambda .NET proporciona la funcionalidad para compilar la aplicación en un contenedor de Docker mediante una imagen AL2023.

Para evitar posibles problemas de compatibilidad entre arquitecturas, le recomendamos encarecidamente que compile el código en un entorno con la misma arquitectura de procesador que configuró para la función. Para obtener más información sobre las limitaciones de la compilación entre arquitecturas, consulte [Compilación entre arquitecturas](#) en la documentación de Microsoft .NET.

## Requisitos previos

### Docker

Para utilizar la AOT nativa, el código de la función debe compilarse en un entorno con el mismo sistema operativo AL2023 que el tiempo de ejecución de .NET 8. Los comandos de la CLI de .NET de las siguientes secciones utilizan Docker para desarrollar y crear funciones de Lambda en un entorno AL2023.

### SDK para .NET 8

La compilación nativa anticipada es una característica de .NET 8. Debe instalar el [SDK para .NET 8](#) en su máquina de compilación, no solo el tiempo de ejecución.

### Amazon.Lambda.Tools

Para crear sus funciones de Lambda, utilice la [extensión de herramientas globales de .NET Amazon.Lambda.Tools](#). Ejecute el siguiente comando para instalar Amazon.Lambda.Tools:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Para obtener más información sobre la Amazon.Lambda.Tools de la extensión de la CLI de .NET, consulte el repositorio en GitHub [Extensiones de AWS para la CLI de .NET](#).

### Amazon.Lambda.Templates

Para generar el código de la función de Lambda, utilice el paquete NuGet [Amazon.Lambda.Templates](#). Para instalar este paquete de plantillas, ejecute el siguiente comando:

```
dotnet new install Amazon.Lambda.Templates
```

## Introducción

Tanto la CLI global de .NET como la AWS Serverless Application Model (AWS SAM) proporcionan plantillas de introducción para crear aplicaciones mediante la AOT nativa. Para crear su primera función de Lambda AOT nativa, lleve a cabo los pasos de las siguientes instrucciones.

Para inicializar e implementar una función de Lambda nativa compilada en AOT

1. Inicialice un nuevo proyecto con la plantilla AOT nativa y, a continuación, navegue hasta el directorio que contiene los archivos `.cs` y `.csproj` creados. En este ejemplo, asignamos un nombre a nuestra función `NativeAotSample`.

```
dotnet new lambda.NativeAOT -n NativeAotSample
cd ./NativeAotSample/src/NativeAotSample
```

El archivo `Function.cs` creado por la plantilla AOT nativa contiene el siguiente código de función.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.RuntimeSupport;
using Amazon.Lambda.Serialization.SystemTextJson;
using System.Text.Json.Serialization;

namespace NativeAotSample;

public class Function
{
    /// <summary>
    /// The main entry point for the Lambda function. The main function is called
    /// once during the Lambda init phase. It
    /// initializes the .NET Lambda runtime client passing in the function handler
    /// to invoke for each Lambda event and
    /// the JSON serializer to use for converting Lambda JSON format to the .NET
    /// types.
    /// </summary>
    private static async Task Main()
    {
        Func<string, ILambdaContext, string> handler = FunctionHandler;
        await LambdaBootstrapBuilder.Create(handler, new
        SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>())
        .Build()
    }
}
```



```
        .RunAsync();
    }

    /// <summary>
    /// A simple function that takes a string and does a ToUpper.
    ///
    /// To use this handler to respond to an AWS event, reference the appropriate
package from
    /// https://github.com/aws/aws-lambda-dotnet#events
    /// and change the string input parameter to the desired event type. When the
event type
    /// is changed, the handler type registered in the main method needs to be
updated and the LambdaFunctionJsonSerializerContext
    /// defined below will need the JsonSerializable updated. If the return type
and event type are different then the
    /// LambdaFunctionJsonSerializerContext must have two JsonSerializable
attributes, one for each type.
    ///
    /// When using Native AOT extra testing with the deployed Lambda functions is
required to ensure
    /// the libraries used in the Lambda function work correctly with Native AOT. If
a runtime
    /// error occurs about missing types or methods the most likely solution will be
to remove references to trim-unsafe
    /// code or configure trimming options. This sample defaults to partial TrimMode
because currently the AWS
    /// SDK for .NET does not support trimming. This will result in a larger
executable size, and still does not
    /// guarantee runtime trimming errors won't be hit.
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public static string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

/// <summary>
/// This class is used to register the input event and return type for the
FunctionHandler method with the System.Text.Json source generator.
/// There must be a JsonSerializable attribute for each type used as the input and
return type or a runtime error will occur
```

```
/// from the JSON serializer unable to find the serialization information for
    unknown types.
/// </summary>
[JsonSerializable(typeof(string))]
public partial class LambdaFunctionJsonSerializerContext : JsonSerializerContext
{
    // By using this partial class derived from JsonSerializerContext, we can
    generate reflection free JSON Serializer code at compile time
    // which can deserialize our class and properties. However, we must attribute
    this class to tell it what types to generate serialization code for.
    // See https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-
    text-json-source-generation
}
```

La AOT nativa compila la aplicación en un único binario nativo. El punto de entrada de ese binario es el método `static Main`. Dentro de `static Main`, se inicia el tiempo de ejecución de Lambda y se configura el método `FunctionHandler`. Como parte del arranque del tiempo de ejecución, se configura un serializador generado en la fuente mediante `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Para implementar la aplicación en Lambda, asegúrese de que Docker se esté ejecutando en su entorno local y ejecute el siguiente comando.

```
dotnet lambda deploy-function
```

Entre bastidores, la CLI global de .NET descarga una imagen de Docker de AL2023 y compila el código de la aplicación dentro de un contenedor en ejecución. El binario compilado se devuelve a su sistema de archivos local antes de implementarlo en Lambda.

3. Pruebe la función al ejecutar el siguiente comando. Reemplace `<FUNCTION_NAME>` por el nombre que eligió para la función en el asistente de implementación.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

La respuesta de la CLI incluye detalles de rendimiento para el arranque en frío (duración de la inicialización) y el tiempo total de ejecución de la invocación de la función.

4. Para eliminar los recursos de AWS que creó siguiendo los pasos anteriores, ejecute el siguiente comando. Reemplace `<FUNCTION_NAME>` por el nombre que eligió para la función en el asistente de implementación. Si elimina los recursos de AWS que ya no utiliza, evitará que se facturen gastos innecesarios en su Cuenta de AWS.

```
dotnet lambda delete-function <FUNCTION_NAME>
```

## Serialización

Para implementar funciones en Lambda mediante la AOT nativa, el código de la función debe usar la [serialización generada en la fuente](#). En lugar de utilizar la reflexión en tiempo de ejecución para recopilar los metadatos necesarios para acceder a las propiedades de los objetos con fines de serialización, los generadores de fuentes generan archivos fuente en C# que se compilan al crear la aplicación. Para configurar correctamente el serializador generado en código fuente, asegúrese de incluir todos los objetos de entrada y salida que utilice su función, así como cualquier tipo personalizado. Por ejemplo, una función de Lambda que recibe eventos de una API de REST de API Gateway y devuelve un tipo de Product personalizado incluiría un serializador definido de la siguiente manera.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]  
[JsonSerializable(typeof(APIGatewayProxyResponse))]  
[JsonSerializable(typeof(Product))]  
public partial class CustomSerializer : JsonSerializerContext  
{  
}
```

## Recorte

La AOT nativa recorta el código de la aplicación como parte de la compilación para garantizar que el binario sea lo más pequeño posible. .NET 8 para Lambda proporciona un mejor soporte de reducción de tamaño en comparación con las versiones anteriores de .NET. Se ha añadido compatibilidad con las [bibliotecas de tiempo de ejecución de Lambda](#), [SDK de .NET de AWS](#), las [anotaciones de Lambda para .NET](#) y .NET 8.

Estas mejoras ofrecen la posibilidad de eliminar las advertencias de recorte durante el tiempo de compilación, pero .NET nunca será completamente seguro para la reducción. Esto significa que las partes de las bibliotecas en las que se basa su función pueden recortarse como parte del paso de compilación. Para gestionarlo, puede definir `TrimmerRootAssemblies` en su archivo `.csproj`, tal como se muestra en el ejemplo siguiente.

```
<ItemGroup>  
  <TrimmerRootAssembly Include="AWSSDK.Core" />  
</ItemGroup>
```

```
<TrimmerRootAssembly Include="AWSXRayRecorder.Core" />
<TrimmerRootAssembly Include="AWSXRayRecorder.Handlers.AwsSdk" />
<TrimmerRootAssembly Include="Amazon.Lambda.APIGatewayEvents" />
<TrimmerRootAssembly Include="bootstrap" />
<TrimmerRootAssembly Include="Shared" />
</ItemGroup>
```

Tenga en cuenta que cuando reciba una advertencia de recorte, es posible que añadir la clase que genera la advertencia a `TrimmerRootAssembly` no resuelva el problema. Una advertencia de recorte indica que la clase está intentando acceder a otra clase que no se puede determinar hasta el tiempo de ejecución. Para evitar errores de tiempo de ejecución, añada esta segunda clase a `TrimmerRootAssembly`.

Para obtener más información sobre la administración de las advertencias de recorte, consulte [Introducción a las advertencias de recorte](#) en la documentación de Microsoft .NET.

## Resolución de problemas

Error: no se admite la compilación nativa entre sistemas operativos.

Su versión de la herramienta global de .NET Core `Amazon.Lambda.Tools` está desactualizada. Actualice la herramienta a la versión más reciente e inténtelo de nuevo.

Docker: el sistema operativo de imágenes "Linux" no se puede utilizar en esta plataforma.

Docker está configurado en su sistema para usar contenedores de Windows. Cambie a contenedores de Linux para ejecutar el entorno de compilación nativa anticipada.

Para obtener más información sobre los errores comunes, consulte el repositorio en GitHub [Enfoque nativo anticipado de AWS para .NET](#).

# Uso del objeto de contexto de Lambda para recuperar información de funciones de C#

Cuando Lambda ejecuta su función, pasa un objeto context al [controlador](#). Este objeto proporciona propiedades con información acerca de la invocación, la función y el entorno de ejecución.

## Propiedades de context

- `FunctionName`: el nombre de la función de Lambda.
- `FunctionVersion`: la [versión](#) de la función.
- `InvokedFunctionArn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `MemoryLimitInMB`: cantidad de memoria asignada a la función.
- `AwsRequestId`: el identificador de la solicitud de invocación.
- `LogGroupName`: grupo de registros de para la función.
- `LogStreamName`: el flujo de registro de la instancia de la función.
- `RemainingTime(TimeSpan)`: el número de milisegundos que quedan antes del tiempo de espera de la ejecución.
- `Identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
- `ClientContext`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.
- `Logger`: el [objeto logger](#) para la función.

Puede utilizar información en el objeto `ILambdaContext` para generar información sobre la invocación de su función con fines de supervisión. El código siguiente proporciona un ejemplo de cómo agregar información de contexto a un marco de registro estructurado. En este ejemplo, la función agrega `AwsRequestId` a los resultados del registro. La función también utiliza la propiedad `RemainingTime` para cancelar una tarea en vuelo si está a punto de agotarse el tiempo de espera de la función de Lambda.

```
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer)  
  
namespace GetProductHandler;
```

```
public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request, ILambdaContext context)
    {
        Logger.AppendKey("AwsRequestId", context.AwsRequestId);

        var id = request.PathParameters["id"];

        using var cts = new CancellationTokenSource();

        try
        {
            cts.CancelAfter(context.RemainingTime.Add(TimeSpan.FromSeconds(-1)));

            var databaseRecord = await this._repo.GetById(id, cts.Token);

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
        finally
        {
            cts.Cancel();

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
    }
}
```

# Registro y supervisión de las funciones de Lambda de C#

AWS Lambda supervisa de forma automática funciones de Lambda y envía entradas de registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación y otros resultados del código de su función al flujo de registro. Para obtener más información acerca de Registros de CloudWatch, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

## Secciones

- [Crear una función que devuelve registros](#)
- [Uso de herramientas y bibliotecas de registro](#)
- [Uso de Powertools para AWS Lambda \(.NET\) y AWS SAM para el registro estructurado](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)

## Crear una función que devuelve registros

Para generar registros desde el código de función, puede utilizar los métodos de [la clase Console](#) o cualquier biblioteca de registro que escriba en `stdout` o en `stderr`.

El tiempo de ejecución de .NET registra las líneas START, END y REPORT de cada invocación. La línea del informe proporciona los siguientes detalles.

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.
- Duración: la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- Duración facturada: la cantidad de tiempo facturado por la invocación.
- Tamaño de memoria: la cantidad de memoria asignada a la función.
- Máximo de memoria usada: la cantidad de memoria utilizada por la función.

- Duración de inicio: para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- TraceId de XRAY: para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- SegmentId: para solicitudes rastreadas, el ID del segmento de X-Ray.
- Muestras: para solicitudes rastreadas, el resultado del muestreo.

## Uso de herramientas y bibliotecas de registro

[Powertools para AWS Lambda \(.NET\)](#) es un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores. La [utilidad de registro](#) proporciona un logger optimizado para Lambda que incluye información adicional sobre el contexto de la función en todas las funciones con un resultado estructurado como JSON. Utilice esta utilidad para hacer lo siguiente:

- Capturar campos clave del contexto de Lambda, arranque en frío y resultados de registro de estructuras como JSON
- Registrar los eventos de invocación de Lambda cuando se le indique (desactivado de forma predeterminada)
- Imprimir todos los registros solo para un porcentaje de las invocaciones mediante el muestreo de registros (desactivado de forma predeterminada)
- Agregar claves adicionales al registro estructurado en cualquier momento
- Utilizar un formateador de registros personalizado (traiga su propio formateador) para generar registros en una estructura compatible con el RFC de registro de su organización

## Uso de Powertools para AWS Lambda (.NET) y AWS SAM para el registro estructurado

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de C# con módulos de [Powertools para AWS Lambda \(.NET\)](#) integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje `hello world`.



## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- .NET 6 o .NET 8
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación utilizando la plantilla de TypeScript de tipo Hola Mundo.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type Zip --runtime dotnet6 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

### Note

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

5. Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Invoque el punto de conexión de la API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

7. Para obtener los registros de la función, ejecute [sam logs](#). Para obtener más información, consulte [Uso de registros](#) en la Guía para desarrolladores de AWS Serverless Application Model.

```
sam logs --stack-name sam-app
```

El resultado del registro tendrá este aspecto:

```
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8
2023-02-20T14:15:27.988000 INIT_START Runtime Version:
dotnet:6.v13 Runtime Version ARN: arn:aws:lambda:ap-
southeast-2::runtime:699f346a05dae24c58c45790bc4089f252bf17dae3997e79b17d939a288aa1ec
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:28.229000
START RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Version: $LATEST
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:29.259000
2023-02-20T14:15:29.201Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"_aws":{"Timestamp":1676902528962,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ColdStart","Unit":"Count"}],"Dimensions":
[["FunctionName"],["Service"]]}]},"FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","Service":"PowertoolsHelloWorld","ColdStart":1}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.479000
2023-02-20T14:15:30.479Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"ColdStart":true,"XrayTraceId":"1-63f3807f-5dbcb9910c96f50742707542","CorrelationId":"d3d
a549-4d67b2fdc015","FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionVersion":"$LATEST","FunctionMemorySize":256,"FunctionArn":"arn:aws:lambda:
southeast-2:123456789012:function:sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionRequestId":"bed25b38-d012-42e7-ba28-
f272535fb80e","Timestamp":"2023-02-20T14:15:30.4602970Z","Level":"Information","Service":"Pow-
ertoolsHelloWorld API - HTTP 200"}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.599000
2023-02-20T14:15:30.599Z bed25b38-d012-42e7-ba28-f272535fb80e info
{"_aws":{"Timestamp":1676902528922,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ApiRequestCount","Unit":"Count"}],"Dimensions":
[["Service"]]}]},"Service":"PowertoolsHelloWorld","ApiRequestCount":1}
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000 END
RequestId: bed25b38-d012-42e7-ba28-f272535fb80e
```

```
2023/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2023-02-20T14:15:30.680000
REPORT RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Duration: 2450.99 ms
  Billed Duration: 2451 ms Memory Size: 256 MB    Max Memory Used: 74 MB  Init
  Duration: 240.05 ms
XRAY TraceId: 1-63f3807f-5dbcb9910c96f50742707542      SegmentId: 16b362cd5f52cba0
```

8. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

## Administración de retención de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros de forma indefinida, elimine el grupo de registros o configure un periodo de retención después del cual CloudWatch los eliminará de forma automática. Para configurar la retención de registros, agregue lo siguiente a la plantilla de AWS SAM:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

### Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

### Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb... ",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en `base64` para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad `base64` está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar `base64 -D`.

Example `get-logs.sh` script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza `sed` para eliminar las comillas del archivo de salida y permanece inactivo

durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando `get-log-events`.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como `get-logs.sh`.

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

```

{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

# Instrumentación de código C# en AWS Lambda

Lambda se integra con AWS X-Ray para permitirle seguir, depurar y optimizar aplicaciones de Lambda. Puede utilizar X-Ray para seguir una solicitud mientras atraviesa los recursos de la aplicación, que pueden incluir funciones de Lambda y otros servicios de AWS.

Para enviar datos de seguimiento a X-Ray, puede utilizar una de estas tres bibliotecas de SDK:

- [AWS Distro for OpenTelemetry \(ADOT\)](#): una distribución segura, lista para producción y con soporte de AWS del OpenTelemetry (OTel) SDK.
- [AWS X-Ray SDK para .NET](#): un SDK para generar y enviar datos de seguimiento a X-Ray.
- [Powertools para AWS Lambda \(.NET\)](#): un kit de herramientas para desarrolladores destinado a implementar prácticas recomendadas sin servidor y aumentar la velocidad de los desarrolladores.

Cada uno de los SDK ofrecen formas de enviar los datos de telemetría al servicio X-Ray. Tras ello, se puede utilizar X-Ray para consultar, filtrar y obtener información sobre las métricas de rendimiento de la aplicación con el fin de identificar problemas y oportunidades de optimización.

## Important

Los SDK de X-Ray y Powertools para AWS Lambda son parte de una solución de instrumentación completamente integrada que ofrece AWS. Las capas Lambda de ADOT forman parte de un estándar que abarca todo el sector para la instrumentación de seguimiento que recopila más datos en general, pero es posible que no sean adecuadas para todos los casos de uso. Puede implementar el seguimiento integral en X-Ray con cualquiera de las soluciones. Para obtener más información sobre cuál elegir, consulte [Elegir entre SDK de AWS Distro para OpenTelemetry y X-Ray](#).

## Secciones

- [Uso de Powertools para AWS Lambda \(.NET\) y AWS SAM para el seguimiento](#)
- [Uso del SDK de X-Ray para instrumentar las funciones de .NET](#)
- [Activación del seguimiento con la consola de Lambda](#)
- [Activación del seguimiento con la API de Lambda](#)
- [Activación del seguimiento con AWS CloudFormation](#)
- [Interpretación de un seguimiento de X-Ray](#)



# Uso de Powertools para AWS Lambda (.NET) y AWS SAM para el seguimiento

Siga los pasos que figuran a continuación para descargar, crear e implementar una aplicación de muestra “Hola, mundo” de C# con módulos de [Powertools para AWS Lambda \(.NET\)](#) integrados mediante AWS SAM. Esta aplicación implementa un backend de API básico y utiliza Powertools para emitir registros, métricas y seguimiento. Consta de un punto de conexión de Amazon API Gateway y una función de Lambda. Cuando se envía una solicitud GET al punto de conexión de API Gateway, la función de Lambda se invoca, envía registros y métricas a CloudWatch mediante Embedded Metric Format y envía seguimiento a AWS X-Ray. La función devuelve el mensaje hello world.

## Requisitos previos

Para completar los pasos de esta sección, debe disponer de lo siguiente:

- .NET 6 o .NET 8
- [Versión 2 de la AWS CLI](#)
- [Versión 1.75 o posterior de la CLI de AWS SAM](#). Si tiene una versión anterior de la CLI de AWS SAM, consulte [Actualización de la CLI de AWS SAM](#).

## Implementar una aplicación de ejemplo de AWS SAM

1. Inicialice la aplicación utilizando la plantilla de TypeScript de tipo Hola Mundo.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type Zip --runtime dotnet6 --no-tracing
```

2. Compile la aplicación.

```
cd sam-app && sam build
```

3. Implemente la aplicación.

```
sam deploy --guided
```

4. Siga las indicaciones que aparecen en pantalla. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima Enter.

**Note**

En HelloWorldFunction es posible que no tenga definida la autorización, ¿está bien?, asegúrese de ingresar y.

- Obtenga la URL de la aplicación implementada:

```
aws cloudformation describe-stacks --stack-name sam-app --query  
'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

- Invoque el punto de conexión de la API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Si se realiza de forma correcta, verá el siguiente resultado:

```
{"message":"hello world"}
```

- Para obtener el seguimiento de la función, ejecute [sam traces](#).

```
sam traces
```

El resultado del seguimiento tendrá este aspecto:

```
New XRay Service Graph  
Start time: 2023-02-20 23:05:16+08:00  
End time: 2023-02-20 23:05:16+08:00  
Reference Id: 0 - AWS::Lambda - sam-app-HelloWorldFunction-pNjujb7mEoew - Edges:  
[1]  
  Summary_statistics:  
    - total requests: 1  
    - ok count(2XX): 1  
    - error count(4XX): 0  
    - fault count(5XX): 0  
    - total response time: 2.814  
Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-pNjujb7mEoew  
- Edges: []  
  Summary_statistics:  
    - total requests: 1  
    - ok count(2XX): 1
```

```

- error count(4XX): 0
- fault count(5XX): 0
- total response time: 2.429
Reference Id: 2 - (Root) AWS::ApiGateway::Stage - sam-app/Prod - Edges: [0]
Summary_statistics:
- total requests: 1
- ok count(2XX): 1
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 2.839
Reference Id: 3 - client - sam-app/Prod - Edges: [2]
Summary_statistics:
- total requests: 0
- ok count(2XX): 0
- error count(4XX): 0
- fault count(5XX): 0
- total response time: 0

XRay Event [revision 3] at (2023-02-20T23:05:16.521000) with id
(1-63f38c2c-270200bf1d292a442c8e8a00) and duration (2.877s)
- 2.839s - sam-app/Prod [HTTP: 200]
- 2.836s - Lambda [HTTP: 200]
- 2.814s - sam-app-HelloWorldFunction-pNjujb7mEoew [HTTP: 200]
- 2.429s - sam-app-HelloWorldFunction-pNjujb7mEoew
- 0.230s - Initialization
- 2.389s - Invocation
- 0.600s - ## FunctionHandler
- 0.517s - Get Calling IP
- 0.039s - Overhead

```

8. Se trata de un punto de conexión de API pública al que se puede acceder a través de Internet. Se recomienda eliminar el punto de conexión después de las pruebas.

```
sam delete
```

X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

## Uso del SDK de X-Ray para instrumentar las funciones de .NET

Puede instrumentar su código de función para registrar metadatos y rastrear llamadas descendentes. Para registrar detalles sobre las llamadas que realiza su función a otros recursos y servicios, use el AWS X-Ray SDK para .NET. Para obtener el SDK, agregue los paquetes `AWSXRayRecorder` al archivo de proyecto.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="2.1.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.7.103.24" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.7.104.3" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.13.0" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.11.0" />
  </ItemGroup>
</Project>
```

Hay una gama de paquetes Nuget que proporcionan instrumentación automática para SDKs AWS, Entity Framework y solicitudes HTTP. Para ver el conjunto completo de opciones de configuración, consulte [SDK AWS X-Ray para .NET](#) en la Guía para desarrolladores AWS X-Ray.

Una vez que haya agregado los paquetes Nuget deseados, configure la instrumentación automática. La mejor práctica es realizar esta configuración fuera de la función de control de la función. Esto le permite aprovechar la reutilización del entorno de ejecución para mejorar el rendimiento de la función. En el siguiente ejemplo de código, se llama al método `RegisterXRayForAllServices` en el constructor de la función para añadir instrumentación a todas las llamadas al SDK de AWS.

```
[assembly:
  LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
{
```

```
private readonly IDatabaseRepository _repo;

public Function()
{
    // Add auto instrumentation for all AWS SDK calls
    // It is important to call this method before initializing any SDK clients
    AWSSDKHandler.RegisterXRayForAllServices();
    this._repo = new DatabaseRepository();
}

public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
{
    var id = request.PathParameters["id"];

    var databaseRecord = await this._repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
}
}
```

## Activación del seguimiento con la consola de Lambda

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

### Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

## Activación del seguimiento con la API de Lambda

Configure el rastreo en la función Lambda con AWS CLI o SDK de AWS, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)
- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Activación del seguimiento con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:
```

```
function:
  Type: AWS::Serverless::Function
  Properties:
    Tracing: Active
    ...
```

## Interpretación de un seguimiento de X-Ray

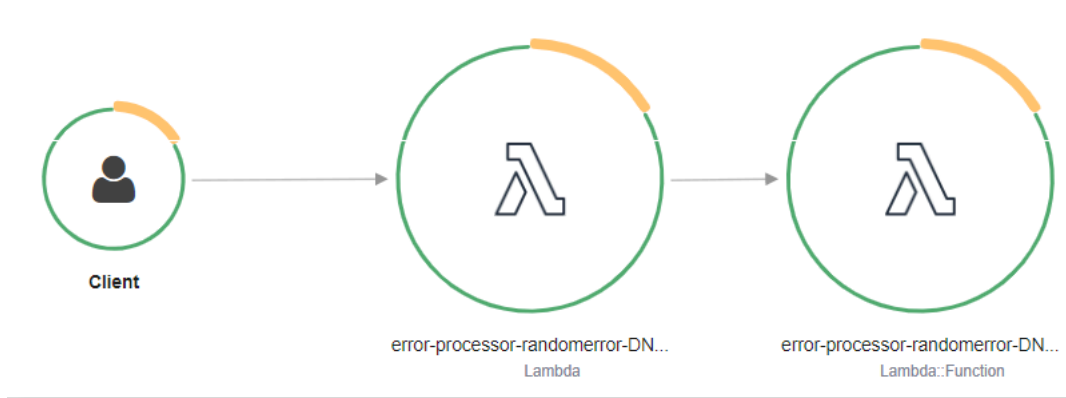
La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

Después de configurar el seguimiento activo, se pueden observar solicitudes específicas a través de la aplicación. El [gráfico de servicios de X-Ray](#) muestra información sobre la aplicación y todos sus componentes. En el siguiente ejemplo, se muestra una aplicación con dos funciones. La función principal procesa eventos y, a veces, devuelve errores. La segunda función de la cadena procesa los errores que aparecen en el primer grupo de registros y utiliza el SDK de AWS para llamar a X-Ray, Amazon Simple Storage Service (Amazon S3) y Registros de Amazon CloudWatch.

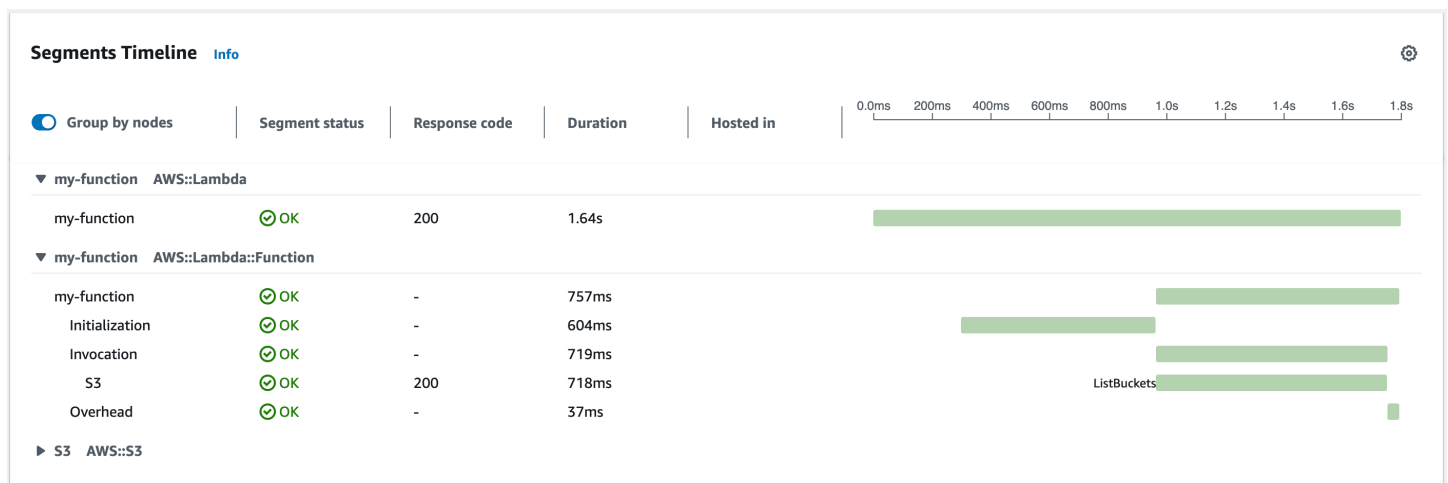


X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica. En el siguiente ejemplo, se muestra un seguimiento con estos dos segmentos. Ambos se denominan my-function, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.



En este ejemplo, el segmento `AWS::Lambda::Function` aparece ampliado para mostrar sus tres subsegmentos.

**Note**

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de



registro del sistema y los segmentos de seguimiento emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

El seguimiento de ejemplo que se muestra aquí ilustra el segmento de función de estilo antiguo. Las diferencias entre los segmentos de estilo antiguo y nuevo se describen en los párrafos siguientes.

Estos cambios se implementarán en las próximas semanas, y todas las funciones en todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de seguimiento.

El segmento de función de estilo antiguo contiene los siguientes subsegmentos:

- **Inicialización:** representa el tiempo dedicado a cargar la función y ejecutar el [código de inicialización](#). Este subsegmento aparece únicamente para el primer evento que procesa cada instancia de la función.
- **Invocación:** representa el tiempo dedicado a ejecutar el código del controlador.
- **Sobrecarga:** representa el tiempo que el tiempo de ejecución de Lambda dedica a prepararse para gestionar el siguiente evento.

El segmento de función de estilo nuevo no contiene ningún subsegmento de `Invocation`. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento de la función. Para obtener más información sobre la estructura de los segmentos de funciones de estilo antiguo y nuevo, consulte [the section called “Comprensión de los rastros”](#).

También puede instrumentar clientes HTTP, registrar consultas SQL y crear subsegmentos personalizados con anotaciones y metadatos. Para obtener más información, consulte [AWS X-Ray SDK para .NET](#) en la Guía para desarrolladores de AWS X-Ray.

### Precios

Puede utilizar el seguimiento de X-Ray de manera gratuita cada mes hasta un límite determinado como parte del nivel Gratuito de AWS. A partir de ese umbral, X-Ray realiza cargos por almacenamiento y recuperación del seguimiento. Para obtener más información, consulte [Precios de AWS X-Ray](#).

# Prueba de funciones de AWS Lambda en C#

## Note

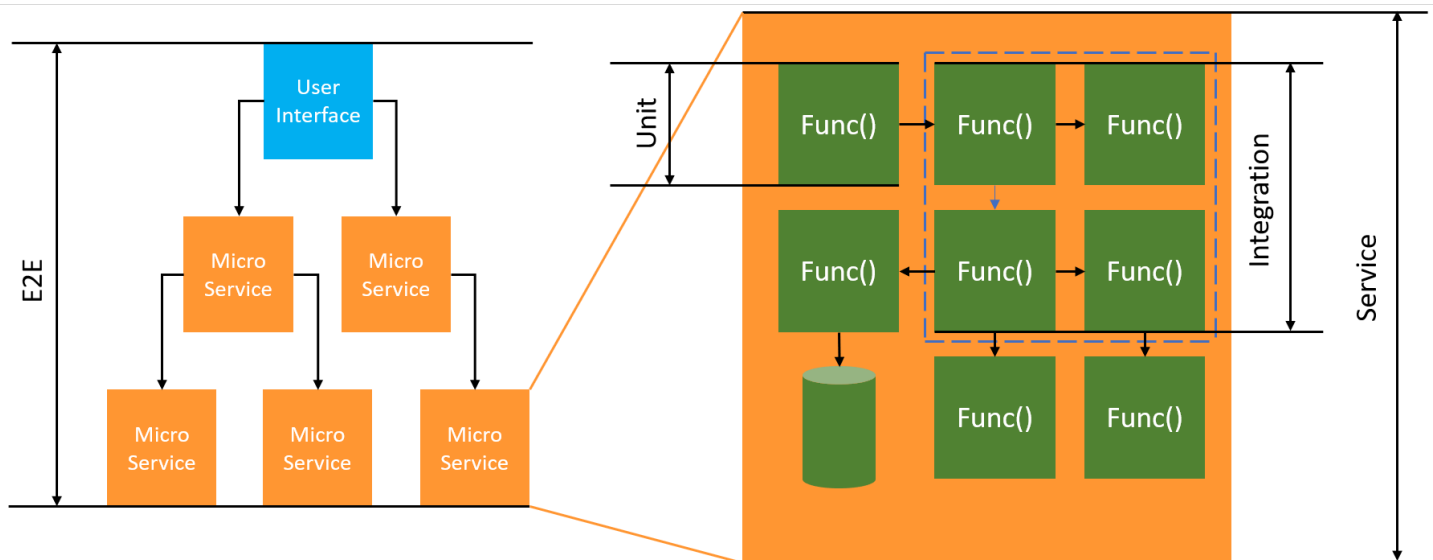
Consulte el capítulo [Prueba de funciones](#) para obtener una introducción completa a las técnicas y prácticas recomendadas para probar soluciones sin servidor.

Para probar las funciones sin servidor, se utilizan tipos y técnicas de prueba tradicionales, pero también se debe considerar la posibilidad de probar las aplicaciones sin servidor en conjunto. Las pruebas basadas en la nube proporcionan la medida más precisa de la calidad tanto de las funciones como de las aplicaciones sin servidor.

Una arquitectura de aplicaciones sin servidor incluye servicios administrados que proporcionan las funcionalidades críticas de las aplicaciones mediante llamadas a la API. Por este motivo, el ciclo de desarrollo debe incluir pruebas automatizadas que verifiquen las funcionalidades cuando la función y los servicios interactúen.

Si no crea pruebas basadas en la nube, pueden surgir problemas debido a las diferencias entre su entorno local y el entorno implementado. El proceso de integración continua debe ejecutar pruebas con un conjunto de recursos aprovisionados en la nube antes de promover el código al siguiente entorno de implementación, como el control de calidad, el ensayo o la producción.

Siga leyendo esta breve guía para obtener información sobre las estrategias de prueba para aplicaciones sin servidor, o visite el [repositorio de ejemplos de pruebas sin servidor](#) a fin de profundizar en ejemplos prácticos y específicos para el lenguaje y el tiempo de ejecución de su elección.



Para las pruebas sin servidor, seguirá escribiendo unidades, integraciones y end-to-end pruebas.

- Pruebas unitarias: pruebas que se ejecutan en un bloque de código aislado. Por ejemplo, verificar la lógica empresarial para calcular los gastos de envío en función de un elemento y un destino determinados.
- Pruebas de integración: pruebas en las que participan dos o más componentes o servicios que interactúan, normalmente en un entorno de nube. Por ejemplo, verificar si una función procesa los eventos de una cola.
- End-to-end Pruebas electrónicas: pruebas que verifican el comportamiento en toda la aplicación. Por ejemplo, garantizar que la infraestructura esté configurada correctamente y que los eventos fluyan entre los servicios tal como se espera para registrar el pedido de un cliente.

## Pruebas de aplicaciones sin servidor

Por lo general, utilizará una combinación de métodos para probar el código de sus aplicaciones sin servidor, como las pruebas en la nube, las pruebas con simulaciones y, en ocasiones, las pruebas con emuladores.

### Pruebas en la nube

Las pruebas en la nube son valiosas para todas las fases de las pruebas, incluidas las pruebas unitarias, las pruebas de integración y end-to-end las pruebas. Las pruebas se ejecutan con el código implementado en la nube y se interactúa con los servicios basados en la nube. Este enfoque proporciona la medida más precisa de la calidad del código.

Una forma práctica de depurar la función de Lambda en la nube es a través de la consola con un evento de prueba. Un evento de prueba es una entrada JSON a su función. Si la función no requiere una entrada, el evento puede ser un documento JSON vacío ({}). La consola proporciona ejemplos de eventos para una variedad de integraciones de servicios. Tras crear un evento en la consola, puede compartirlo con su equipo para que las pruebas sean más sencillas y coherentes.

### Note

[Probar una función en la consola](#) es una forma rápida de empezar, pero la automatización de los ciclos de prueba garantiza la calidad de la aplicación y la velocidad de desarrollo.

## Herramientas para pruebas

Para acelerar su ciclo de desarrollo, hay una serie de herramientas y técnicas que puede utilizar al probar sus funciones. Por ejemplo, tanto [Accelerate AWS SAM](#) como el [modo de vigilancia de AWS CDK](#) reducen el tiempo necesario para actualizar los entornos de nube.

La forma en que se define el código de la función de Lambda facilita la adición de pruebas unitarias. Lambda requiere un constructor público sin parámetros para inicializar la clase. La introducción de un segundo constructor interno le permite controlar las dependencias que utiliza su aplicación.

```
[assembly:
  LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(): this(null)
    {
    }

    internal Function(IDatabaseRepository repo)
    {
        this._repo = repo ?? new DatabaseRepository();
    }
}
```

```
public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
request)
{
    var id = request.PathParameters["id"];

    var databaseRecord = await this._repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
    };
}
}
```

Para escribir una prueba para esta función, puede inicializar una nueva instancia de su clase `Function` y pasar una implementación simulada de `IDatabaseRepository`. Los siguientes ejemplos utilizan `XUnit`, `Moq` y `FluentAssertions` para escribir una prueba sencilla que garantice que `FunctionHandler` devuelve un código de estado 200.

```
using Xunit;
using Moq;
using FluentAssertions;

public class FunctionTests
{
    [Fact]
    public async Task TestLambdaHandler_WhenInputIsValid_ShouldReturn200StatusCode()
    {
        // Arrange
        var mockDatabaseRepository = new Mock<IDatabaseRepository>();

        var functionUnderTest = new Function(mockDatabaseRepository.Object);

        // Act
        var response = await functionUnderTest.FunctionHandler(new
APIGatewayProxyRequest());

        // Assert
        response.StatusCode.Should().Be(200);
    }
}
```

---

Para ver ejemplos más detallados, incluidos ejemplos de pruebas asíncronas, consulte el repositorio de [ejemplos de pruebas de.NET](#) en. GitHub

# Creación de funciones de Lambda con PowerShell

En las siguientes secciones se explica cómo se aplican los conceptos fundamentales y los patrones de programación comunes al crear código de funciones de Lambda en PowerShell.

Lambda proporciona las siguientes aplicaciones de muestra para PowerShell:

- [blank-powershell](#): una función de PowerShell que muestra el uso de registro, las variables de entorno y el AWS SDK .

Antes de comenzar, primero debe configurar un entorno de desarrollo de PowerShell. Para obtener instrucciones al respecto, consulte [Configuración de un entorno PowerShell de desarrollo](#).

Para obtener más información sobre cómo utilizar el módulo AWSLambdaPSCore para descargar proyectos de muestra de PowerShell a partir de plantillas, cómo crear paquetes de implementación de PowerShell y cómo implementar funciones de PowerShell en la nube de AWS, consulte [Implemente funciones PowerShell Lambda con archivos de archivos.zip](#).

Lambda proporciona los siguientes tiempos de ejecución para lenguajes .NET:

.NET

Nombre	Identificador	Sistema operativo	Fecha de baja	Bloqueo de la función Crear	Bloqueo de la función Actualizar
.NET 8	dotnet8	Amazon Linux 2023			
.NET 6	dotnet6	Amazon Linux 2	20 de diciembre de 2024	28 de febrero de 2025	31 de marzo de 2025

Temas

- [Configuración de un entorno PowerShell de desarrollo](#)
- [Implemente funciones PowerShell Lambda con archivos de archivos.zip](#)
- [Definir el controlador de las funciones de Lambda en PowerShell](#)

- [Uso del objeto de contexto de Lambda para recuperar la información de las funciones de PowerShell](#)
- [Registro y supervisión de las funciones de Lambda de Powershell](#)



## Configuración de un entorno PowerShell de desarrollo

Lambda proporciona un conjunto de herramientas y bibliotecas para el PowerShell tiempo de ejecución. Para obtener instrucciones de instalación, consulte [Herramientas Lambda para PowerShell](#) más información. [GitHub](#)

El AWSLambdaPSCore módulo incluye los siguientes cmdlets para ayudar a crear y publicar funciones Lambda PowerShell :

- `Get-AWSPowerShellLambdaTemplate` — Devuelve una lista de plantillas de introducción.
- `Nuevo-AWSPowerShellLambda` — Crea un PowerShell script inicial basado en una plantilla.
- `PublicarAWSPowerShellLambda`: publica un PowerShell script determinado en Lambda.
- `NuevoAWSPowerShellLambdaPackage`: crea un paquete de implementación Lambda que puede usar en un sistema de CI/CD para la implementación.

# Implemente funciones PowerShell Lambda con archivos de `archivos.zip`

Un paquete de despliegue para el PowerShell tiempo de ejecución contiene el PowerShell script, PowerShell los módulos necesarios para el PowerShell script y los ensamblajes necesarios para alojar PowerShell Core.

## Creación de la función de Lambda

Para empezar a escribir e invocar un PowerShell script con Lambda, puede usar `New-AWSPowerShellLambda` el cmdlet para crear un script de inicio basado en una plantilla. Puede utilizar el cmdlet `Publish-AWSPowerShellLambda` para implementar su script en Lambda. A continuación, puede probar el script ya sea a través de la línea de comandos o de la consola de Lambda.

Para crear un PowerShell script nuevo, cargarlo y probarlo, haga lo siguiente:

1. Para ver la lista de plantillas disponibles, ejecute el siguiente comando:

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template          Description
-----          -
Basic             Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
...
```

2. Para crear un script de ejemplo basado en la plantilla `Basic`, ejecute el siguiente comando:

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

Se creará un nuevo archivo denominado `MyFirstPSScript.ps1` en un nuevo subdirectorio del directorio actual. El nombre del directorio se basa en el parámetro `-ScriptName`. Puede utilizar el parámetro `-Directory` para elegir otro directorio.

Puede ver que el nuevo archivo tiene el siguiente contenido:

```
# PowerShell script file to run as a Lambda function
#
# When executing in Lambda the following variables are predefined.
```

```
# $LambdaInput - A PSObject that contains the Lambda function input data.
# $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
information about the currently running Lambda environment.
#
# The last item in the PowerShell pipeline is returned as the result of the Lambda
function.
#
# To include PowerShell modules with your Lambda function, like the
AWSPowerShell.NetCore module, add a "#Requires" statement
# indicating the module and version.

#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}

# Uncomment to send the input to CloudWatch Logs
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

3. Para ver cómo se envían los mensajes de registro de tu PowerShell script a Amazon CloudWatch Logs, quita el comentario de la `Write-Host` línea del script de muestra.

Para demostrar cómo puede devolver los datos de sus funciones de Lambda, agregue una nueva línea al final del script con `$PSVersionTable`. Esto añade el `$PSVersionTable` a la PowerShell canalización. Una vez finalizado el PowerShell script, el último objeto de la PowerShell canalización son los datos de retorno de la función Lambda. `$PSVersionTable` es una variable PowerShell global que también proporciona información sobre el entorno de ejecución.

Después de realizar estos cambios, las dos últimas líneas del script de muestra tienen un aspecto similar al siguiente:

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
$PSVersionTable
```

4. Después de editar el archivo `MyFirstPSScript.ps1`, cambie el directorio a la ubicación del script. Después, ejecute el siguiente comando para publicar el script en Lambda:

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name
MyFirstPSScript -Region us-east-2
```

Tenga en cuenta que el parámetro `-Name` especifica el nombre de la función de , que aparece en la consola de Lambda. Puede utilizar esta función para invocar manualmente a su script.

5. Invoque su función con el comando `invoke` AWS Command Line Interface (AWS CLI).

```
> aws lambda invoke --function-name MyFirstPSScript out
```

# Definir el controlador de las funciones de Lambda en PowerShell

Cuando se invoca una función de Lambda, el controlador de Lambda invoca al script de PowerShell.

Cuando el script de PowerShell se invoca, se predefinen las siguientes variables:

- ***\$LambdaInput***: un PObject que contiene la entrada al controlador. Esta entrada puede estar formada por datos de eventos (publicados por un origen de eventos) o por una entrada personalizada que proporcione, como una cadena o cualquier objeto de datos personalizado.
- ***\$LambdaContext***: un objeto Amazon.Lambda.Core.ILambdaContext que puede utilizar para obtener acceso a información sobre la invocación actual, como el nombre de la función actual, el límite de memoria, el tiempo de ejecución restante y el registro.

Por ejemplo, fíjese en el siguiente código de ejemplo de PowerShell.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}  
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Este script devuelve la propiedad `FunctionName` que obtiene de la variable `$LambdaContext`.

## Note

Es necesario que utilice la instrucción `#Requires` dentro de sus scripts de PowerShell para indicar los módulos de los que dependen los scripts. Esta instrucción realiza dos tareas importantes. 1) Comunica a otros desarrolladores los módulos que utiliza el script; y 2) identifica los módulos dependientes que necesitan las herramientas de AWS PowerShell para incluir en el paquete con el script, como parte de la implementación. Para obtener más información sobre la instrucción `#Requires` en PowerShell, consulte [Acerca de los requisitos](#). Para obtener más información sobre los paquetes de implementación de PowerShell, consulte [Implemente funciones PowerShell Lambda con archivos de archivos.zip](#).

Cuando su función de Lambda para PowerShell utiliza los cmdlets de AWS PowerShell, asegúrese de establecer una instrucción `#Requires` que haga referencia al módulo `AWSPowerShell.NetCore`, que admite PowerShell Core y no el módulo `AWSPowerShell`, que solo admite Windows PowerShell. Además, asegúrese de utilizar la versión 3.3.270.0 o posterior de `AWSPowerShell.NetCore`, que optimiza el proceso de importación del

cmdlet. Si utiliza una versión anterior, experimentará unos arranques en frío más largos. Para obtener más información, consulte [AWS Tools for PowerShell](#).

## Devolución de datos

Algunas invocaciones Lambda están destinadas a devolver los datos a la persona que llama. Por ejemplo, si una invocación se hizo en respuesta a una solicitud web procedente de API Gateway, entonces nuestra función de Lambda debe devolver la respuesta. Para PowerShell Lambda, el último objeto que se añade a la canalización de PowerShell son los datos de devolución de la invocación de Lambda. Si el objeto es una cadena, los datos se devuelven tal cual. De lo contrario, el objeto se convierte en JSON mediante el uso del cmdlet `ConvertTo-Json`.

Por ejemplo, observe la siguiente instrucción de PowerShell, que añade `$PSVersionTable` a la canalización de PowerShell:

```
$PSVersionTable
```

Una vez que el script de PowerShell ha finalizado, el último objeto en la canalización de PowerShell son los datos de devolución de la función de Lambda. `$PSVersionTable` es una variable global de PowerShell que también proporciona información sobre el entorno en ejecución.

# Uso del objeto de contexto de Lambda para recuperar la información de las funciones de PowerShell

Cuando Lambda ejecuta su función, pasa información de contexto haciendo que una variable `$LambdaContext` esté disponible para el [controlador](#). Esta variable proporciona métodos y propiedades con información acerca de la invocación, la función y el entorno de ejecución.

## Propiedades de context

- `FunctionName`: el nombre de la función de Lambda.
- `FunctionVersion`: la [versión](#) de la función.
- `InvokedFunctionArn`: el nombre de recurso de Amazon (ARN) que se utiliza para invocar esta función. Indica si el invocador especificó un número de versión o alias.
- `MemoryLimitInMB`: cantidad de memoria asignada a la función.
- `AwsRequestId`: el identificador de la solicitud de invocación.
- `LogGroupName`: grupo de registros de para la función.
- `LogStreamName`: el flujo de registro de la instancia de la función.
- `RemainingTime`: el número de milisegundos que quedan antes del tiempo de espera de la ejecución.
- `Identity`: (aplicaciones móviles) Información acerca de la identidad de Amazon Cognito que autorizó la solicitud.
- `ClientContext`: (aplicaciones móviles) Contexto de cliente proporcionado a Lambda por la aplicación cliente.
- `Logger`: el [objeto logger](#) para la función.

El siguiente fragmento de código PowerShell muestra una función de controlador sencilla que imprime parte de la información de contexto.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

# Registro y supervisión de las funciones de Lambda de Powershell

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre y envía registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación al flujo de registro y retransmite los registros y otras salidas desde el código de la función. Para obtener más información, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#).

Esta página describe cómo producir resultados de registro a partir del código de la función de Lambda o registros de acceso mediante AWS Command Line Interface, la consola de Lambda o la consola de CloudWatch.

## Secciones

- [Crear una función que devuelve registros](#)
- [Visualización de los registros en la consola de Lambda](#)
- [Visualización de los registros de en la consola de CloudWatch](#)
- [Visualización de los registros mediante la AWS Command Line Interface \(AWS CLI\)](#)
- [Eliminación de registros](#)

## Crear una función que devuelve registros

Para generar registros desde el código de su función, puede usar cmdlets en [Microsoft.PowerShell.Utility](#) o en cualquier módulo de registro que escriba en `stdout` o en `stderr`. El siguiente ejemplo utiliza `Write-Host`.

Example [function/Handler.ps1](#): registro

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host `## Environment variables
Write-Host AWS_LAMBDA_FUNCTION_VERSION=$Env:AWS_LAMBDA_FUNCTION_VERSION
Write-Host AWS_LAMBDA_LOG_GROUP_NAME=$Env:AWS_LAMBDA_LOG_GROUP_NAME
Write-Host AWS_LAMBDA_LOG_STREAM_NAME=$Env:AWS_LAMBDA_LOG_STREAM_NAME
Write-Host AWS_EXECUTION_ENV=$Env:AWS_EXECUTION_ENV
Write-Host AWS_LAMBDA_FUNCTION_NAME=$Env:AWS_LAMBDA_FUNCTION_NAME
Write-Host PATH=$Env:PATH
Write-Host `## Event
```



```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 3)
```

## Example formato de registro

```
START RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Version: $LATEST
Importing module ./Modules/AWSPowerShell.NetCore/3.3.618.0/AWSPowerShell.NetCore.psd1
[Information] - ## Environment variables
[Information] - AWS_LAMBDA_FUNCTION_VERSION=$LATEST
[Information] - AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/blank-powershell-
function-18CIXMPLHFAJJ
[Information] - AWS_LAMBDA_LOG_STREAM_NAME=2020/04/01/
[$LATEST]53c5xmpl52d64ed3a744724d9c201089
[Information] - AWS_EXECUTION_ENV=AWS_Lambda_dotnet6_powershell_1.0.0
[Information] - AWS_LAMBDA_FUNCTION_NAME=blank-powershell-function-18CIXMPLHFAJJ
[Information] - PATH=/var/lang/bin:/usr/local/bin:/usr/bin/./bin:/opt/bin
[Information] - ## Event
[Information] -
{
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1523232000000",
        "SenderId": "123456789012",
        "ApproximateFirstReceiveTimestamp": "1523232000001"
      },
      ...
    }
  ]
}
END RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed
REPORT RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Duration: 3906.38 ms Billed
Duration: 4000 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19
ms
XRAY TraceId: 1-5e843da6-733cxmple7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled:
true
```

El tiempo de ejecución de .NET registra las líneas START, END y REPORT de cada invocación. La línea del informe proporciona los siguientes detalles.

### Campos de datos de línea REPORT

- RequestId: el ID de solicitud único para la invocación.

- **Duración:** la cantidad de tiempo que el método de controlador de función pasó procesando el evento.
- **Duración facturada:** la cantidad de tiempo facturado por la invocación.
- **Tamaño de memoria:** la cantidad de memoria asignada a la función.
- **Máximo de memoria usada:** la cantidad de memoria utilizada por la función.
- **Duración de inicio:** para la primera solicitud servida, la cantidad de tiempo que tardó el tiempo de ejecución en cargar la función y ejecutar código fuera del método del controlador.
- **TraceId de XRAY:** para las solicitudes rastreadas, el [ID de seguimiento de AWS X-Ray](#).
- **SegmentId:** para solicitudes rastreadas, el ID del segmento de X-Ray.
- **Muestras:** para solicitudes rastreadas, el resultado del muestreo.

## Visualización de los registros en la consola de Lambda

Puede utilizar la consola de Lambda para ver la salida del registro después de invocar una función de Lambda.

Si su código se puede probar desde el editor de código integrado, encontrará los registros en los resultados de ejecución. Cuando utilice la característica de prueba de la consola para invocar una función, encontrará la Salida de registro en la sección de Detalles.

## Visualización de los registros de en la consola de CloudWatch

Puede utilizar la consola Amazon CloudWatch para ver los registros de todas las invocaciones de funciones de Lambda.

### Visualización de los registros en la consola CloudWatch

1. En la consola de CloudWatch, abra la [página de grupos de registro](#).
2. Seleccione el grupo de registros para su función (`/aws/lambda/your-function-name`).
3. Elija una secuencia de registro.

Cada flujo de registro se corresponde con una [instancia de su función](#). Aparece un flujo de registro cuando actualiza la función de Lambda y cuando se crean instancias adicionales para manejar varias invocaciones simultáneas. Para encontrar registros para una invocación específica, le recomendamos que interfiera su función con AWS X-Ray. X-Ray registra los detalles sobre la solicitud y el flujo de registro en el seguimiento.

## Visualización de los registros mediante la AWS Command Line Interface (AWS CLI)

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRi0C1mMTU0LTExZTgt0GNkYS0y0Tc0YzVlNGZiMjEgVmVyc2lvc21vb... ",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad `base64` para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad `base64` está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar `base64 -D`.

Example `get-logs.sh` script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza `sed` para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando `get-log-events`.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como `get-logs.sh`.

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

## Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```
{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
```

```
        "timestamp": 1559763003218,
        "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
        "ingestionTime": 1559763018353
    },
    {
        "timestamp": 1559763003218,
        "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

## Eliminación de registros

Los grupos de registro no se eliminan automáticamente cuando se elimina una función. Para evitar almacenar registros indefinidamente, elimine el grupo de registros o [configure un periodo de retención](#) después de lo cual los registros se eliminan automáticamente.

# Creación de funciones de Lambda con Rust

Como Rust compila en código nativo, no necesita un tiempo de ejecución dedicado para ejecutar el código de Rust en Lambda. En su lugar, utilice el [cliente de tiempo de ejecución de Rust](#) para crear su proyecto localmente y, a continuación, impleméntelo en Lambda con el tiempo de ejecución `provided.al2023` o `provided.al2`. Cuando usa `provided.al2023` o `provided.al2`, Lambda mantiene actualizado el sistema operativo automáticamente con los parches más recientes.

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

## Herramientas y bibliotecas para Rust

- [AWS SDK para Rust](#): el SDK de AWS para Rust proporciona las API de Rust para interactuar con los servicios de infraestructura de Amazon Web Services.
- [Cliente de tiempo de ejecución de Rust para Lambda](#): el cliente de tiempo de ejecución de Rust es un paquete experimental. Está sujeto a cambios importantes y no se recomienda usarlo en la producción.
- [Cargo Lambda](#): esta biblioteca proporciona una aplicación de línea de comandos para trabajar con funciones de Lambda creadas con Rust.
- [Lambda HTTP](#): esta biblioteca proporciona un contenedor para trabajar con eventos HTTP.
- [Extensión de Lambda](#): esta biblioteca proporciona asistencia para escribir extensiones de Lambda con Rust.
- [Eventos de AWS Lambda](#): esta biblioteca proporciona definiciones de tipos de integraciones de orígenes de eventos comunes.

## Aplicaciones de Lambda para Rust de muestra

- [Función de Lambda básica](#): una función de Rust que muestra cómo procesar eventos básicos.
- [Función de Lambda con gestión de errores](#): una función de Rust que muestra cómo gestionar los errores de Rust personalizados en Lambda.

- [Función de Lambda con recursos compartidos](#): un proyecto de Rust que inicializa los recursos compartidos antes de crear la función de Lambda.
- [Eventos de Lambda HTTP](#): una función de Rust que gestiona eventos HTTP.
- [Eventos de Lambda HTTP con encabezados CORS](#): una función de Rust que usa Tower para inyectar encabezados CORS.
- [API de REST Lambda](#): una API de REST que usa Axum y Diesel para conectarse a una base de datos de PostgreSQL.
- [Demostración de Rust sin servidor](#): un proyecto de Rust que muestra el uso de las bibliotecas de Rust de Lambda, el registro, las variables de entorno y el SDK de AWS.
- [Extensión de Lambda básica](#): una extensión de Rust que muestra cómo procesar eventos de extensión básicos.
- [Extensión de Amazon Data Firehose para registros de Lambda](#): una extensión de Rust que muestra cómo enviar los registros de Lambda a Firehose.

## Temas

- [Definir el controlador de funciones de Lambda en Rust](#)
- [Uso del objeto de contexto Lambda para recuperar la información de la función de Rust](#)
- [Procesamiento de eventos HTTP con Rust](#)
- [Implementar funciones de Rust Lambda con archivos .zip](#)
- [Registro y supervisión de las funciones de Lambda en Rust](#)



# Definir el controlador de funciones de Lambda en Rust

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

El controlador de la función de Lambda es el método del código de la función que procesa eventos. Cuando se invoca una función, Lambda ejecuta el método del controlador. La función se ejecuta hasta que el controlador devuelve una respuesta, se cierra o se agota el tiempo de espera.

## Temas

- [Conceptos básicos del controlador de Rust](#)
- [Uso del estado compartido](#)
- [Prácticas recomendadas de codificación para las funciones de Lambda en Rust](#)

## Conceptos básicos del controlador de Rust

Escriba el código de la función de Lambda como un ejecutable de Rust. Implemente el código de función del controlador y una función principal e incluya lo siguiente:

- La caja [lambda\\_runtime](#) de crates.io, que implementa el modelo de programación de Lambda para Rust.
- Incluya [Tokio](#) en las dependencias. El [cliente de tiempo de ejecución de Rust para Lambda](#) utiliza Tokio para gestionar llamadas asíncronas.

## Example — Controlador de Rust que procesa eventos JSON

El siguiente ejemplo utiliza la caja [serde\\_json](#) para procesar eventos JSON básicos:

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}
```

```
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

Tenga en cuenta lo siguiente:

- `use`: importa las bibliotecas que necesita la función de Lambda.
- `async fn main`: el punto de entrada que ejecuta el código de la función de Lambda. El cliente de tiempo de ejecución de Rust utiliza [Tokio](#) como un tiempo de ejecución asíncrono, por lo que debe anotar la función principal con `#[tokio::main]`.
- `async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error>`: esta es la firma del controlador de Lambda. Incluye el código que se ejecuta cuando se invoca la función.
  - `LambdaEvent<Value>`: este es un tipo genérico que describe el evento recibido por el tiempo de ejecución de Lambda, así como el [contexto de la función de Lambda](#).
  - `Result<Value, Error>`: la función devuelve un tipo `Result`. Si la función se ejecuta correctamente, el resultado es un valor JSON. Si la función no se ejecuta correctamente, el resultado es un error.

## Uso del estado compartido

Es posible declarar variables compartidas independientes del código de controlador de la función de Lambda. Estas variables pueden ayudarlo a cargar la información de estado durante [Fase "init"](#), antes de que la función reciba cualquier evento.

Example — Compartir el cliente de Amazon S3 entre instancias de funciones

Tenga en cuenta lo siguiente:

- `use aws_sdk_s3::Client`: este ejemplo requiere que agregue `aws-sdk-s3 = "0.26.0"` a la lista de dependencias del archivo `Cargo.toml`.
- `aws_config::from_env`: este ejemplo requiere que agregue `aws-config = "0.55.1"` a la lista de dependencias del archivo `Cargo.toml`.

```
use aws_sdk_s3::Client;
```

```

use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde::{Deserialize, Serialize};

#[derive(Deserialize)]
struct Request {
    bucket: String,
}

#[derive(Serialize)]
struct Response {
    keys: Vec<String>,
}

async fn handler(client: &Client, event: LambdaEvent<Request>) -> Result<Response,
Error> {
    let bucket = event.payload.bucket;
    let objects = client.list_objects_v2().bucket(bucket).send().await?;
    let keys = objects
        .contents()
        .map(|s| s.iter().flat_map(|o| o.key().map(String::from)).collect())
        .unwrap_or_default();
    Ok(Response { keys })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    let shared_config = aws_config::from_env().load().await;
    let client = Client::new(&shared_config);
    let shared_client = &client;
    lambda_runtime::run(service_fn(move |event: LambdaEvent<Request>| async move {
        handler(&shared_client, event).await
    })))
    .await
}

```

## Prácticas recomendadas de codificación para las funciones de Lambda en Rust

Siga las directrices de la siguiente lista para utilizar las prácticas recomendadas de codificación al crear sus funciones de Lambda:

- Separe el controlador de Lambda de la lógica del núcleo. Esto le permite probar las distintas unidades de la función con mayor facilidad.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el [entorno de ejecución](#).
- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación.
- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.
- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API

internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.

- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

# Uso del objeto de contexto Lambda para recuperar la información de la función de Rust

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

Cuando Lambda ejecuta su función, agrega un objeto de contexto al `LambdaEvent` que recibe el [controlador](#). Este objeto proporciona propiedades con información acerca de la invocación, la función y el entorno de ejecución.

## Propiedades de context

- `request_id`: el ID de solicitud de AWS generado por el servicio de Lambda.
- `deadline`: la fecha límite de ejecución de la invocación actual en milisegundos.
- `invoked_function_arn`: el nombre de recurso de Amazon (ARN) de la función de Lambda que se invoca.
- `xray_trace_id`: el ID de seguimiento AWS X-Ray de la invocación actual.
- `client_content`: el objeto de contexto del cliente enviado por el SDK de AWS móvil. Este campo está vacío a menos que la función se invoque mediante un SDK de AWS móvil.
- `identity`: la identidad de Amazon Cognito que invocó la función. Este campo está vacío a menos que la solicitud de invocación a las API de Lambda se haya realizado con credenciales de AWS emitidas por los grupos de identidades de Amazon Cognito.
- `env_config`: la configuración de la función de Lambda de las variables de entorno local. Esta propiedad incluye información como el nombre de la función, la asignación de memoria, la versión y los flujos de registro.

## Acceso a la información del contexto de invocación

Las funciones de Lambda tienen acceso a metadatos acerca de su entorno y la solicitud de invocación. El objeto `LambdaEvent` que recibe el controlador de funciones incluye los metadatos `context`:

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
```

```
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    let invoked_function_arn = event.context.invoked_function_arn;
    Ok(json!({ "message": format!("Hello, this is function
{invoked_function_arn}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

# Procesamiento de eventos HTTP con Rust

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

Las API de Amazon API Gateway, los equilibradores de carga de aplicación y las [URL de la función de Lambda](#) pueden enviar eventos HTTP a Lambda. Puede utilizar la caja [aws\\_lambda\\_events](#) de crates.io para procesar eventos de estos orígenes.

## Example — Gestionar la solicitud de proxy de API Gateway

Tenga en cuenta lo siguiente:

- use `aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse}`: la caja [aws\\_lambda\\_events](#) incluye muchos eventos de Lambda. Para reducir el tiempo de compilación, utilice indicadores de funciones para activar los eventos que necesite. Ejemplo: `aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- use `http::HeaderMap`: esta importación requiere que agregue la caja [http](#) a sus dependencias.

```
use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse};
use http::HeaderMap;
use lambda_runtime::{service_fn, Error, LambdaEvent};

async fn handler(
    _event: LambdaEvent<ApiGatewayProxyRequest>,
) -> Result<ApiGatewayProxyResponse, Error> {
    let mut headers = HeaderMap::new();
    headers.insert("content-type", "text/html".parse().unwrap());
    let resp = ApiGatewayProxyResponse {
        status_code: 200,
        multi_value_headers: headers.clone(),
        is_base64_encoded: false,
        body: Some("Hello AWS Lambda HTTP request".into()),
        headers,
    };
};
```



```

    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}

```

El [cliente de tiempo de ejecución de Rust para Lambda](#) también proporciona una abstracción sobre estos tipos de eventos que permite trabajar con tipos de HTTP nativos, independientemente del servicio que envíe los eventos. El siguiente código es equivalente al ejemplo anterior y funciona de forma inmediata con las URL de las funciones de Lambda, los equilibradores de carga de aplicación y API Gateway.

### Note

La caja [lambda\\_http](#) usa la caja [lambda\\_runtime](#) que se encuentra debajo. No es necesario que importe `lambda_runtime` por separado.

## Example — Gestionar solicitudes HTTP

```

use lambda_http::{service_fn, Error, IntoResponse, Request, RequestExt, Response};

async fn handler(event: Request) -> Result<impl IntoResponse, Error> {
    let resp = Response::builder()
        .status(200)
        .header("content-type", "text/html")
        .body("Hello AWS Lambda HTTP request")
        .map_err(Box::new)?;
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_http::run(service_fn(handler)).await
}

```

Para obtener otro ejemplo sobre cómo usar `lambda_http`, consulte el [ejemplo de código http-axum](#) en el repositorio de GitHub de AWS Labs.

## Ejemplos de eventos HTTP Lambda para Rust

- [Eventos de Lambda HTTP](#): una función de Rust que gestiona eventos HTTP.
- [Eventos de Lambda HTTP con encabezados CORS](#): una función de Rust que usa Tower para inyectar encabezados CORS.
- [Eventos HTTP Lambda con recursos compartidos](#): una función de Rust que usa recursos compartidos inicializados antes de crear el controlador de funciones.

# Implementar funciones de Rust Lambda con archivos .zip

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

En esta página se describe cómo compilar la función de Rust y luego implementar el binario compilado a AWS Lambda con [Cargo Lambda](#). También muestra cómo implementar el binario compilado con la AWS Command Line Interface y la CLI de AWS Serverless Application Model.

## Secciones

- [Requisitos previos](#)
- [Creación de funciones de Rust en macOS, Windows o Linux](#)
- [Implementación del binario de la función de Rust con Cargo Lambda](#)
- [Invocación de la función de Rust con Cargo Lambda](#)

## Requisitos previos

- [Rust](#)
- [AWS Command Line Interface \(AWS CLI\) versión 2](#)

## Creación de funciones de Rust en macOS, Windows o Linux

Los siguientes pasos muestran cómo crear el proyecto para su primera función de Lambda con Rust y cómo compilarlo con [Cargo Lambda](#).

1. Instale Cargo Lambda, un subcomando de Cargo que compila las funciones de Rust para Lambda en macOS, Windows y Linux.

Para instalar Cargo Lambda en cualquier sistema que tenga instalado Python 3, use pip:

```
pip3 install cargo-lambda
```

Para instalar Cargo Lambda en macOS o Linux, use Homebrew:

```
brew tap cargo-lambda/cargo-lambda
brew install cargo-lambda
```

Para instalar Cargo Lambda en Windows, use [Scoop](#):

```
scoop bucket add cargo-lambda
scoop install cargo-lambda/cargo-lambda
```

Para ver otras opciones, consulte [Instalación](#) en la documentación de Cargo Lambda.

2. Cree la estructura del paquete. Este comando crea un código de función básico en `src/main.rs`. Puede usar este código para realizar pruebas o sustituirlo por su propio código.

```
cargo lambda new my-function
```

3. Dentro del directorio raíz del paquete, ejecute el subcomando [build](#) para compilar el código de la función.

```
cargo lambda build --release
```

(Opcional) Si desea usar AWS Graviton2 en Lambda, agregue el indicador `--arm64` para compilar el código para las CPU ARM.

```
cargo lambda build --release --arm64
```

4. Antes de implementar la función de Rust, configure las credenciales de AWS en su equipo.

```
aws configure
```

## Implementación del binario de la función de Rust con Cargo Lambda

Utilice el subcomando [deploy](#) para implementar el binario compilado en Lambda. Este comando crea un [rol de ejecución](#) y luego crea la función de Lambda. Para especificar un rol de ejecución existente, utilice el [indicador --iam-role](#).

```
cargo lambda deploy my-function
```

## Implementación del binario de la función de Rust con la AWS CLI

También puede implementar el binario con la AWS CLI.

1. Para crear el paquete de despliegue .zip, utilice el subcomando [build](#).

```
cargo lambda build --release --output-format zip
```

2. Ejecute el comando [create-function](#) para implementar el paquete .zip en Lambda.
  - En `--runtime`, especifique `provided.al2023`. Este es un [tiempo de ejecución exclusivo del sistema operativo](#). Los tiempos de ejecución exclusivos del sistema operativo se utilizan para implementar binarios compilados y tiempos de ejecución personalizados en Lambda.
  - En `--role`, especifique el ARN del [rol de ejecución](#).

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime provided.al2023 \  
  --role arn:aws:iam::111122223333:role/lambda-role \  
  --handler rust.handler \  
  --zip-file fileb://target/lambda/my-function/bootstrap.zip
```

## Implementación del binario de la función de Rust con la CLI de AWS SAM

También puede implementar el binario con la CLI de AWS SAM.

1. Cree una plantilla de AWS SAM con la definición del recurso y la propiedad. En `Runtime`, especifique `provided.al2023`. Este es un [tiempo de ejecución exclusivo del sistema operativo](#). Los tiempos de ejecución exclusivos del sistema operativo se utilizan para implementar binarios compilados y tiempos de ejecución personalizados en Lambda.

Para obtener más información sobre la implementación de funciones de Lambda mediante AWS SAM, consulte [AWS::Serverless::Function](#) en la Guía para desarrolladores de AWS Serverless Application Model.

Example Definición de recursos y propiedades de SAM para un binario de Rust

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
Description: SAM template for Rust binaries
Resources:
  RustFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: target/lambda/my-function/
      Handler: rust.handler
      Runtime: provided.al2023
Outputs:
  RustFunction:
    Description: "Lambda Function ARN"
    Value: !GetAtt RustFunction.Arn
```

2. Utilice el subcomando [build](#) para compilar la función.

```
cargo lambda build --release
```

3. Utilice el comando [sam deploy](#) para implementar la función en Lambda.

```
sam deploy --guided
```

Para obtener más información sobre cómo crear funciones de Rust con la CLI de AWS SAM, consulte [Creación de funciones de Rust Lambda con Cargo Lambda](#) en la Guía para desarrolladores de AWS Serverless Application Model.

## Invocación de la función de Rust con Cargo Lambda

Use el subcomando [invoke](#) para probar la función con una carga.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

## Invocación de la función de Rust con la AWS CLI

También puede usar la AWS CLI para invocar la función.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --payload '{"command": "Hello world"}' /tmp/out.txt
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

# Registro y supervisión de las funciones de Lambda en Rust

## Note

El [cliente de tiempo de ejecución de Rust](#) es un paquete experimental. Está sujeto a cambios y destinado únicamente para fines de evaluación.

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre y envía registros a Amazon CloudWatch. Su función de Lambda viene con un grupo de registros de Registros de CloudWatch y con un flujo de registro para cada instancia de su función. El entorno de tiempo de ejecución de Lambda envía detalles sobre cada invocación al flujo de registro y retransmite los registros y otras salidas desde el código de la función. Para obtener más información, consulte [Uso de Registros de Amazon CloudWatch con AWS Lambda](#). Esta página describe cómo producir resultados de registro a partir del código de la función de Lambda.

## Creación de una función que escribe registros

Para generar registros desde el código de su función, puede utilizar cualquier función de registro que escriba en `stdout` o `stderr`, como el macro `println!`. El siguiente ejemplo utiliza `println!` para imprimir un mensaje cuando se inicia el controlador de funciones y antes de que finalice.

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    println!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    println!("Rust function responds to {}", &first_name);
    Ok(json!({ "message": format!("Hello, {}!", first_name) }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```



## Implementación del registro avanzado con la caja Tracing

[Tracing](#) es un marco para instrumentar los programas de Rust a fin de recopilar información de diagnóstico estructurada y basada en eventos. Este marco proporciona utilidades para personalizar los niveles y formatos de resultado del registro, como la creación de mensajes de registro JSON estructurados. Para utilizar este marco, debe inicializar un `subscriber` antes de implementar el controlador de funciones. A continuación, puede utilizar macros de rastreo como `debug`, `info` y `error` para especificar el nivel de registro que desea para cada situación.

### Example — Uso de la caja Tracing

Tenga en cuenta lo siguiente:

- `tracing_subscriber::fmt().json()`: cuando se incluye esta opción, los registros están en formato JSON. Para usar esta opción, debe incluir la función `json` en la dependencia `tracing-subscriber` (por ejemplo, `tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]`: esta anotación genera un intervalo cada vez que se invoca el controlador. El intervalo agrega el ID de la solicitud a cada línea de registro.
- `{ %first_name }`: este constructo agrega el campo `first_name` a la línea de registro donde se usa. El valor de este campo corresponde a la variable con el mismo nombre.

```
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    tracing::info!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    tracing::info!({ %first_name }, "Rust function responds to event");
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt().json()
        .with_max_level(tracing::Level::INFO)
        // this needs to be set to remove duplicated information in the log.
}
```

```
.with_current_span(false)
// this needs to be set to false, otherwise ANSI color codes will
// show up in a confusing manner in CloudWatch logs.
.with_ansi(false)
// disabling time is handy because CloudWatch will add the ingestion time.
.without_time()
// remove the name of the function from every log entry
.with_target(false)
.init();
lambda_runtime::run(service_fn(handler)).await
}
```

Cuando se invoca esta función de Rust, imprime dos líneas de registro similares a las siguientes:

```
{"level":"INFO","fields":{"message":"Rust function invoked"},"spans":
[{"req_id":"45daaaa7-1a72-470c-9a62-e79860044bb5","name":"handler"}]}
{"level":"INFO","fields":{"message":"Rust function responds to
event","first_name":"David"},"spans":[{"req_id":"45daaaa7-1a72-470c-9a62-
e79860044bb5","name":"handler"}]}
```

# Prácticas recomendadas para trabajar con funciones AWS Lambda

A continuación se indican las prácticas recomendadas para utilizar AWS Lambda:

## Temas

- [Código de función](#)
- [Función de configuración](#)
- [Escalabilidad de las funciones](#)
- [Métricas y alarmas](#)
- [Uso de secuencias](#)
- [Prácticas recomendadas de seguridad](#)

## Código de función

- Reutilice el entorno de ejecución para mejorar el rendimiento de la función. Inicialice los clientes de SDK y las conexiones de base de datos fuera del controlador de funciones y almacene localmente en caché los recursos estáticos en el directorio /tmp. Las invocaciones posteriores procesadas por la misma instancia de su función pueden reutilizar estos recursos. Esto ahorra costes al reducir el tiempo de ejecución de la función.

Para evitar posibles filtraciones de datos entre las invocaciones, no utilice el entorno de ejecución para almacenar datos de usuario, eventos u otra información con implicaciones de seguridad. Si su función se basa en un estado mutable que no se puede almacenar en la memoria dentro del controlador, considere crear una función independiente o versiones independientes de una función para cada usuario.

- Utilice una directiva keep-alive para mantener conexiones persistentes. Lambda purga las conexiones inactivas a lo largo del tiempo. Si intenta reutilizar una conexión inactiva al invocar una función, se producirá un error de conexión. Para mantener la conexión persistente, use la directiva keep-alive asociada al tiempo de ejecución. Para ver un ejemplo, consulte [Reutilización de conexiones con Keep-Alive en Node.js](#).
- Utilice [variables de entorno](#) para pasar parámetros operativos a su función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrelo como una variable de entorno.

- Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados. Si lo hace accidentalmente, establezca la simultaneidad reservada de funciones en 0 para limitar inmediatamente todas las invocaciones de la función mientras actualiza el código.
- No utilice API no documentadas y no públicas en el código de la función de Lambda. Para tiempos de ejecución administrados de AWS Lambda, Lambda aplica periódicamente actualizaciones funcionales y de seguridad a las API internas de Lambda. Estas actualizaciones de las API internas pueden ser incompatibles con versiones anteriores, lo que conlleva consecuencias no deseadas, como errores de invocación si su función depende de estas API no públicas. Consulte la [referencia de la API](#) para obtener una lista de las API disponibles públicamente.
- Escriba el código idempotente. Escribir el código idempotente para las funciones garantiza que los eventos duplicados se gestionen de la misma manera. El código debe validar y gestionar correctamente los eventos duplicados. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#).

Para conocer las prácticas recomendadas de código para idiomas específicos, consulte las siguientes secciones:

- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Node.js”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Typescript”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Python”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Ruby”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Go”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en C#”](#)
- [the section called “Prácticas recomendadas de codificación para las funciones de Lambda en Rust”](#)

## Función de configuración

- La prueba de desempeño de la función de Lambda es una parte crucial a la hora de garantizar la elección de la configuración de memoria óptima. Cualquier aumento del tamaño de la memoria causa un aumento equivalente de la CPU disponible para la función. El uso de la memoria de la función se determina por invocación y puede visualizarse en [Amazon CloudWatch](#). En cada invocación se genera una entrada REPORT :, tal como se muestra a continuación:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed  
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Analice el campo Max Memory Used: para determinar si la función necesita más memoria o si ha aprovisionado en exceso el tamaño de la memoria para la función.

Para encontrar la configuración de memoria adecuada para sus funciones, recomendamos utilizar el proyecto AWS Lambda Power Tuning de código abierto. Para obtener más información, consulte [AWS LambdaAjuste de energía](#) en GitHub.

Para optimizar el rendimiento de las funciones, también recomendamos implementar bibliotecas que puedan aprovechar Advanced Vector Extensions 2 (AVX2). Esto le permite procesar cargas de trabajo exigentes, incluyendo inferencias de machine learning, procesamiento de medios, computación de alto rendimiento (HPC), simulaciones científicas y modelado financiero. Para obtener más información, consulte [Creación de funciones de AWS Lambda más rápidas con AVX2](#).

- Realice una prueba de carga de la función de Lambda para determinar un valor de tiempo de espera óptimo. Es importante analizar cuánto tiempo se ejecuta la función para determinar más fácilmente los posibles problemas con un servicio de dependencias que puedan aumentar la concurrencia de la función más allá de lo esperado. Esto es especialmente importante cuando la función de Lambda realiza llamadas de red a recursos que pueden no gestionar el escalado de Lambda.
- Utilice los permisos más restrictivos al establecer las políticas de IAM. Conozca los recursos y las operaciones que la función de Lambda necesita, y limite el rol de ejecución a estos permisos. Para obtener más información, consulte [Administrar permisos en AWS Lambda](#).
- Familiarícese con los [Cuotas de Lambda](#). A menudo, el tamaño de carga, los descriptores de archivo y el espacio de /tmp se pasan por alto a la hora de determinar los límites de recursos del tiempo de ejecución.

- Elimine las funciones de Lambda que ya no utilice. Al hacerlo, las funciones no utilizadas no contarán innecesariamente para el límite de tamaño del paquete de implementación.
- Si utiliza Amazon Simple Queue Service como origen de eventos, asegúrese de que el valor del tiempo de invocación que se espera para la función no sea superior al valor de [Tiempo de espera de visibilidad](#) de la cola. Esto se aplica tanto a [CreateFunction](#) como a [UpdateFunctionConfiguration](#).
  - En el caso de CreateFunction, AWS Lambda no realizará el proceso de creación de función.
  - En el caso de UpdateFunctionConfiguration, podría dar lugar a invocaciones duplicadas de la función.

## Escalabilidad de las funciones

- Familiarícese con las limitaciones de rendimiento ascendentes y descendentes. Si bien las funciones de Lambda se escalan sin problemas con la carga, es posible que las dependencias ascendentes y descendentes no tengan las mismas capacidades de rendimiento. Si necesita limitar cuán alto se puede escalar su función, puede [configurar la simultaneidad reservada](#) de su función.
- Tolerancia de limitación incorporada. Si su función sincrónica sufre una limitación debido a que el tráfico supera la velocidad de escalado de Lambda, puede utilizar las siguientes estrategias para mejorar la tolerancia a la aceleración:
  - Utilice [Tiempos de espera, reintentos y retroceso con fluctuación](#). La implementación de estas estrategias facilita las invocaciones reintentadas y ayuda a garantizar que Lambda se pueda escalar verticalmente en cuestión de segundos para minimizar las limitaciones de los usuarios finales.
  - Utilice la [simultaneidad aprovisionada](#). La simultaneidad aprovisionada es la cantidad de entornos de ejecución preinicializados que desea asignar a la función. Lambda gestiona las solicitudes entrantes mediante la simultaneidad aprovisionada cuando está disponible. Lambda también puede escalar su función más allá de su configuración de simultaneidad aprovisionada si es necesario. La configuración de la simultaneidad aprovisionada genera cargos adicionales para su cuenta de AWS.

## Métricas y alarmas

- Utilice [Ver las métricas de funciones de Lambda](#) y [Alarmas de CloudWatch](#) en lugar de crear o actualizar una métrica desde dentro de la función de código de Lambda. Es una forma mucho más eficaz de realizar el seguimiento del estado de las funciones de Lambda, y le permitirá identificar los problemas al comienzo del proceso de desarrollo. Por ejemplo, puede configurar una alarma en función de la duración esperada de la invocación de la función de Lambda para solucionar los cuellos de botella o las latencias atribuibles al código de la función.
- Utilice la biblioteca de registro y las [AWS Lambda dimensiones y métricas](#) para detectar los errores de las aplicaciones (por ejemplo, ERR, ERROR, WARNING, etc.)
- Use [Detección de anomalías de costos de AWS](#) para detectar actividad inusual en su cuenta. La detección de anomalías de costos utiliza machine learning para monitorear de forma continua el costo y el uso y, al mismo tiempo, minimizar las alertas positivas falsas. La detección de anomalías de costo utiliza datos de AWS Cost Explorer, que tiene un retraso de hasta 24 horas. Como resultado de ello, puede tardar hasta 24 horas en detectar una anomalía después de que se produce el uso. Para comenzar a utilizar la detección de anomalías de costo, primero debe [registrarse en Cost Explorer](#). Luego, acceda a [Cost Anomaly Detection](#) (Detección de anomalías de costos).

## Uso de secuencias

- Pruebe con diferentes tamaños de lote y de registro para que la frecuencia de sondeo de cada origen de eventos se ajuste a la velocidad con la que la función es capaz de completar su tarea. El parámetro de BatchSize [CreateEventSourceMapping](#) controla el número máximo de registros que se pueden enviar a la función en cada invocación. A menudo, un tamaño de lote mayor puede absorber de forma más eficiente el tráfico adicional asociado a un conjunto de registros mayor, mejorando el desempeño.

De forma predeterminada, Lambda invoca su función tan pronto como los registros estén disponibles. Si el lote que Lambda lee del origen de eventos solo tiene un registro, Lambda envía solo un registro a la función. Para evitar invocar la función con un número de registros pequeño, puede indicar al origen de eventos que almacene en búfer registros durante hasta 5 minutos configurando un plazo de procesamiento por lotes. Antes de invocar la función, Lambda continúa leyendo los registros del origen de eventos hasta que haya recopilado un lote completo, venza el plazo de procesamiento por lotes o el lote alcance el límite de carga de 6 MB. Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

**⚠ Warning**

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

- Aumente el rendimiento del procesamiento de transmisiones de Kinesis añadiendo particiones. Un flujo de Kinesis se compone de una o más particiones. Lambda sondeará cada partición que tenga como máximo una invocación simultánea. Por ejemplo, si su flujo tiene 100 particiones activas, habrá como máximo 100 invocaciones a la función de Lambda ejecutándose simultáneamente. Al aumentar el número de particiones, aumentará directamente el número máximo de invocaciones simultáneas a la función de Lambda, y también puede incrementarse el rendimiento del procesamiento de flujos de Kinesis. Si aumenta el número de particiones de un flujo de Kinesis, asegúrese de que ha elegido una buena clave de partición (consulte [Claves de partición](#)) para los datos, de forma que los registros relacionados acaben en las mismas particiones y los datos estén bien distribuidos.
- Utilice [Amazon CloudWatch](#) en `IteratorAge` para determinar si el flujo de Kinesis se está procesando. Por ejemplo, configure una alarma de CloudWatch con un valor máximo de 30 000 (30 segundos).

## Prácticas recomendadas de seguridad

- Monitoree el uso de AWS Lambda en relación con las mejores prácticas de seguridad con AWS Security Hub. Security Hub utiliza controles de seguridad para evaluar las configuraciones de los recursos y los estándares de seguridad para ayudarlo a cumplir con varios marcos de conformidad. Para obtener más información sobre el uso de Security Hub para evaluar los recursos de Lambda, consulte [controles de AWS Lambda](#) en la Guía del usuario de AWS Security Hub.
- Supervise los registros de actividad de red de Lambda con Amazon GuardDuty Lambda Protection. La protección de GuardDuty Lambda le ayuda a identificar posibles amenazas de seguridad cuando invoca sus funciones de Lambda en su Cuenta de AWS. Por ejemplo, si una de sus funciones consulta una dirección IP asociada con la actividad relacionada con criptomonedas. GuardDuty supervisa los registros de la actividad de la red generados mediante la invocación de



---

funciones de Lambda. Para obtener más información, consulte [Lambda Protection](#) en la Guía del usuario de Amazon GuardDuty.

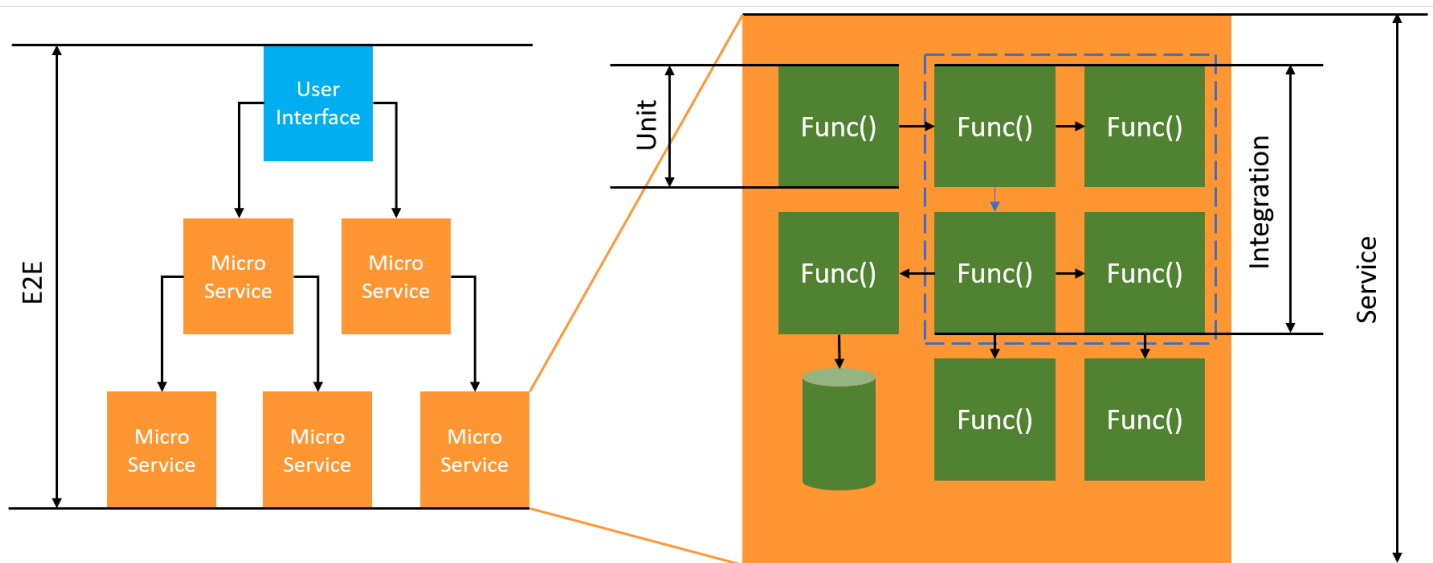
# Cómo realizar pruebas de funciones y aplicaciones sin servidor

Para probar las funciones sin servidor, se utilizan tipos y técnicas de prueba tradicionales, pero también se debe considerar la posibilidad de probar las aplicaciones sin servidor en conjunto. Las pruebas basadas en la nube proporcionan la medida más precisa de la calidad tanto de las funciones como de las aplicaciones sin servidor.

Una arquitectura de aplicaciones sin servidor incluye servicios administrados que proporcionan las funcionalidades críticas de las aplicaciones mediante llamadas a la API. Por este motivo, el ciclo de desarrollo debe incluir pruebas automatizadas que verifiquen las funcionalidades cuando la función y los servicios interactúen.

Si no crea pruebas basadas en la nube, pueden surgir problemas debido a las diferencias entre su entorno local y el entorno implementado. El proceso de integración continua debe ejecutar pruebas con un conjunto de recursos provisionados en la nube antes de promover el código al siguiente entorno de implementación, como el control de calidad, el ensayo o la producción.

Siga leyendo esta breve guía para obtener información sobre las estrategias de prueba para aplicaciones sin servidor, o visite el [repositorio de ejemplos de pruebas sin servidor](#) a fin de profundizar en ejemplos prácticos y específicos para el lenguaje y el tiempo de ejecución de su elección.



Para las pruebas sin servidor, seguirá escribiendo pruebas unitarias, de integración e integrales.

- **Pruebas unitarias:** pruebas que se ejecutan en un bloque de código aislado. Por ejemplo, verificar la lógica empresarial para calcular los gastos de envío en función de un elemento y un destino determinados.
- **Pruebas de integración:** pruebas en las que participan dos o más componentes o servicios que interactúan, normalmente en un entorno de nube. Por ejemplo, verificar si una función procesa los eventos de una cola.
- **Pruebas integrales:** pruebas que verifican el comportamiento de toda una aplicación. Por ejemplo, garantizar que la infraestructura esté configurada correctamente y que los eventos fluyan entre los servicios tal como se espera para registrar el pedido de un cliente.

## Resultados empresariales específicos

Probar soluciones sin servidor puede requerir un poco más de tiempo para configurar las pruebas que verifiquen las interacciones entre los servicios impulsados por eventos. Tenga en cuenta los siguientes motivos empresariales prácticos al leer esta guía:

- Aumente la calidad de una aplicación
- Reduzca el tiempo necesario para crear características y corregir errores

La calidad de una aplicación depende de probar una variedad de escenarios para verificar su funcionalidad. Considerar detenidamente los escenarios empresariales y automatizar esas pruebas para que se ejecuten en los servicios en la nube mejorará la calidad de su aplicación.

Los errores de software y los problemas de configuración afectan mucho menos el costo y la programación cuando se detectan durante un ciclo de desarrollo iterativo. Si los problemas no se detectan durante el desarrollo, más personas tienen que esforzarse el doble para encontrar y solucionar problemas durante el proceso de producción.

Una estrategia de pruebas sin servidor bien planificada aumentará la calidad del software y mejorará el tiempo de iteración al verificar que las funciones de Lambda y las aplicaciones funcionen según lo esperado en un entorno de nube.

## Qué se debe probar

Recomendamos adoptar una estrategia de pruebas que pruebe los comportamientos de los servicios administrados, la configuración de la nube, las políticas de seguridad y la integración con el código para mejorar la calidad del software. Las pruebas de comportamiento, también conocidas como

pruebas de caja negra, verifican que un sistema funcione como se espera sin conocer todos los aspectos internos.

- Ejecute pruebas unitarias para comprobar la lógica empresarial dentro de las funciones de Lambda.
- Verifique que los servicios integrados estén realmente invocados y que los parámetros de entrada sean correctos.
- Compruebe que un evento pase por todos los servicios esperados de principio a fin en un flujo de trabajo.

En la arquitectura tradicional basada en servidores, los equipos suelen definir un ámbito de prueba para incluir únicamente el código que se ejecuta en el servidor de aplicaciones. Otros componentes, servicios o dependencias a menudo se consideran externos y quedan fuera del ámbito de las pruebas.

Las aplicaciones sin servidor suelen constar de pequeñas unidades de trabajo, como las funciones de Lambda que recuperan productos de una base de datos, procesan elementos de una cola o cambian el tamaño de una imagen almacenada. Cada componente se ejecuta en su propio entorno. Es probable que los equipos sean responsables de muchas de estas pequeñas unidades dentro de una sola aplicación.

Algunas funcionalidades de la aplicación pueden delegarse por completo a servicios administrados, como Amazon S3, o crearse sin utilizar ningún código desarrollado internamente. No es necesario probar estos servicios administrados, pero sí la integración con ellos.

## Cómo efectuar las pruebas sin servidor

Es probable que esté familiarizado con la forma de probar aplicaciones implementadas de manera local: escribe pruebas que se ejecutan con código que se ejecuta por completo en el sistema operativo de escritorio o dentro de contenedores. Por ejemplo, puede invocar un componente de servicio web local con una solicitud y, a continuación, hacer afirmaciones sobre la respuesta.

Las soluciones sin servidor se crean a partir del código de funciones y de los servicios administrados basados en la nube, como colas, bases de datos, buses de eventos y sistemas de mensajería. Todos estos componentes están conectados a través de una arquitectura basada en eventos, en la que los mensajes, denominados eventos, fluyen de un recurso a otro. Estas interacciones pueden ser sincrónicas, por ejemplo, cuando un servicio web devuelve resultados de forma inmediata, o una

acción asincrónica que se completa más adelante, como colocar elementos en una cola o iniciar un paso de un flujo de trabajo. Su estrategia de pruebas debe incluir ambos escenarios y probar las interacciones entre los servicios. En el caso de las interacciones asincrónicas, es posible que necesite detectar efectos secundarios en los componentes posteriores que tal vez no se puedan observar de inmediato.

No es práctico replicar un entorno de nube completo, incluidas las colas, las tablas de bases de datos, los buses de eventos, las políticas de seguridad y más. Inevitablemente, encontrará problemas debido a las diferencias entre su entorno local y los entornos implementados en la nube. Las variaciones entre los entornos aumentarán el tiempo de reproducción y corrección de errores.

En las aplicaciones sin servidor, los componentes de la arquitectura suelen existir completamente en la nube, por lo que es necesario realizar pruebas con el código y los servicios de la nube para desarrollar características y corregir errores.

## Técnicas de pruebas

En la práctica, es probable que su estrategia de pruebas incluya una combinación de técnicas para aumentar la calidad de sus soluciones. Utilizará pruebas interactivas rápidas para depurar las funciones en la consola y pruebas unitarias automatizadas para comprobar la lógica empresarial aislada, verificará las llamadas a servicios externos con simulaciones y realizará pruebas ocasionales con emuladores que imitan un servicio.

- Pruebas en la nube: implementa la infraestructura y el código para realizar pruebas con servicios reales, políticas de seguridad, configuraciones y parámetros específicos de la infraestructura. Las pruebas basadas en la nube proporcionan la medida más precisa de la calidad del código.

Depurar una función en la consola es una forma rápida de realizar pruebas en la nube. Puede elegir entre una biblioteca de ejemplos de eventos de prueba o crear un evento personalizado para probar una función de forma aislada. También puede compartir los eventos de prueba a través de la consola con su equipo.

Para automatizar las pruebas durante el ciclo de vida de desarrollo y compilación, tendrá que realizar las pruebas fuera de la consola. Consulte las secciones de pruebas específicas para cada lenguaje de esta guía para conocer las estrategias y los recursos de automatización.

- Pruebas con simulaciones (también denominadas falsificaciones): las simulaciones son objetos del código que simulan y sustituyen a un servicio externo. Las simulaciones proporcionan un comportamiento predefinido para verificar las llamadas de servicio y los parámetros. Una

falsificación es un despliegue simulado que utiliza atajos para simplificar o mejorar el rendimiento. Por ejemplo, un objeto de acceso a datos falso puede devolver datos de un almacén de datos en memoria. Las simulaciones pueden imitar y simplificar dependencias complejas, pero también pueden generar más simulaciones para reemplazar las dependencias anidadas.

- Pruebas con emuladores: puede configurar aplicaciones (a veces de terceros) para que imiten un servicio en la nube en su entorno local. La velocidad es su punto fuerte, pero la configuración y la paridad con los servicios de producción son su punto débil. Use los emuladores con moderación.

## Pruebas en la nube

Las pruebas en la nube son valiosas para todas las fases de las pruebas, incluidas las pruebas unitarias, las pruebas de integración y las pruebas integrales. Cuando realice pruebas con código basado en la nube que también interactúa con los servicios basados en la nube, obtiene la medida más precisa de la calidad del código.

Una forma práctica de ejecutar una función de Lambda en la nube es con un evento de prueba en la AWS Management Console. Un evento de prueba es una entrada JSON a su función. Si la función no requiere una entrada, el evento puede ser un documento JSON vacío ( {} ). La consola proporciona ejemplos de eventos para una variedad de integraciones de servicios. Después de crear un evento en la consola, también puede compartirlo con su equipo para que las pruebas sean más sencillas y coherentes.

Aprenda a [depurar una función de ejemplo en la consola](#).

### Note

Si bien ejecutar funciones en la consola es una forma rápida de depurar, automatizar los ciclos de prueba es fundamental para aumentar la calidad de la aplicación y la velocidad de desarrollo.

Los ejemplos de automatización de pruebas están disponibles en el [repositorio de ejemplos de pruebas sin servidor](#). La siguiente línea de comandos ejecuta un [ejemplo de prueba de integración automática de Python](#):

```
python -m pytest -s tests/integration -v
```

Aunque la prueba se ejecuta de forma local, interactúa con los recursos basados en la nube. Estos recursos se han implementado mediante AWS Serverless Application Model y la herramienta

de la línea de comandos de AWS SAM. El código de prueba recupera primero las salidas de la pila implementada, que incluyen el punto de conexión de la API, el ARN de la función y el rol de seguridad. A continuación, la prueba envía una solicitud al punto de conexión de la API, que responde con una lista de buckets de Amazon S3. Esta prueba se ejecuta en su totalidad con recursos basados en la nube para verificar que esos recursos estén implementados, protegidos y funcionen según lo esperado.

```
===== test session starts =====
platform darwin -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0
-- /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-lambda/
venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-
lambda
plugins: mock-3.10.0
collected 1 item

tests/integration/test_api_gateway.py::TestApiGateway::test_api_gateway

--> Stack outputs:

HelloWorldApi
= https://p7teqs3162.execute-api.us-west-2.amazonaws.com/Prod/hello/
> API Gateway endpoint URL for Prod stage for Hello World function

PythonTestDemo
= arn:aws:lambda:us-west-2:1234567890:function:testing-apigw-lambda-
PythonTestDemo-iSij8evaTdxl
> Hello World Lambda Function ARN

PythonTestDemoIamRole
= arn:aws:iam::1234567890:role/testing-apigw-lambda-PythonTestDemoRole-
IZELQQ9MG4HQ
> Implicit IAM Role created for Hello World function

--> Found API endpoint for "testing-apigw-lambda" stack...
--> https://p7teqs3162.execute-api.us-west-2.amazonaws.com/Prod/hello/
API Gateway response:
amplify-dev-123456789-deployment|myapp-prod-p-loggingbucket-123456|s3-java-
bucket-123456789
PASSED
```

```
===== 1 passed in 1.53s =====
```

Para el desarrollo de aplicaciones nativas en la nube, las pruebas en la nube ofrecen las siguientes ventajas:

- Puede probar todos los servicios disponibles.
- Siempre se utilizan las API de servicio y los valores devueltos más recientes.
- Un entorno de pruebas en la nube se parece mucho a su entorno de producción.
- Las pruebas pueden abarcar políticas de seguridad, cuotas de servicio, configuraciones y parámetros específicos de la infraestructura.
- Cada desarrollador puede crear rápidamente uno o más entornos de prueba en la nube.
- Las pruebas en la nube aumentan la confianza en que el código se ejecutará correctamente en producción.

Las pruebas en la nube tienen algunas desventajas. El aspecto negativo más obvio de las pruebas en la nube es que las implementaciones en entornos de nube suelen tardar más que las implementaciones en entornos de escritorio locales.

Por suerte, herramientas como el [AWS Serverless Application Model \(AWS SAM\) Accelerate](#), el [modo de vigilancia de AWS Cloud Development Kit \(AWSCDK\)](#) y [SST](#) (de terceros) reducen la latencia relacionada con las iteraciones de implementación en la nube. Estas herramientas pueden monitorear su infraestructura y código e implementar automáticamente actualizaciones progresivas en su entorno de nube.

#### Note

Consulte cómo [crear infraestructura como código](#) en la Guía para desarrolladores sin servidor a fin de obtener más información sobre AWS Serverless Application Model, AWS CloudFormation y AWS Cloud Development Kit (AWS CDK).

A diferencia de las pruebas locales, las pruebas en la nube requieren recursos adicionales, lo que puede generar costos de servicio. La creación de entornos de prueba aislados puede aumentar la carga de trabajo de sus equipos de DevOps, especialmente en organizaciones con controles estrictos sobre las cuentas y la infraestructura. Aun así, cuando se trabaja con escenarios de infraestructura complejos, el costo en cuanto al tiempo de los desarrolladores para configurar y



mantener un entorno local complejo puede ser similar (o más costoso) al de utilizar entornos de prueba desechables creados con herramientas de automatización de infraestructura como código.

Las pruebas en la nube, incluso teniendo en cuenta estas consideraciones, siguen siendo la mejor manera de garantizar la calidad de sus soluciones sin servidor.

## Pruebas con simulaciones

Las pruebas con simulaciones son una técnica en la que se crean objetos de reemplazo en el código para simular el comportamiento de un servicio en la nube.

Por ejemplo, puede escribir una prueba que utilice una simulación del servicio Amazon S3 que devuelva una respuesta específica cada vez que se llame al método `CreateObject`. Cuando se ejecuta una prueba, el simulacro devuelve la respuesta programada sin llamar a Amazon S3 ni a ningún otro punto de conexión del servicio.

Los objetos simulados suelen generarse mediante un marco simulado para reducir el esfuerzo de desarrollo. Algunos marcos simulados son genéricos y otros están diseñados específicamente para los AWS SDK, como [Moto](#), una biblioteca de Python para simular servicios y recursos de AWS.

Tenga en cuenta que los objetos simulados se diferencian de los emuladores en que las simulaciones suelen ser creadas o configuradas por un desarrollador como parte del código de prueba, mientras que los emuladores son aplicaciones independientes que exponen la funcionalidad de la misma manera que los sistemas que emulan.

Entre las ventajas de utilizar simulaciones se incluyen las siguientes:

- Las simulaciones pueden simular servicios de terceros que escapan al control de su aplicación, como las API y los proveedores de software como servicio (SaaS), sin necesidad de tener acceso directo a esos servicios.
- Las simulaciones son útiles para probar las condiciones de error, especialmente cuando dichas condiciones son difíciles de simular, como una interrupción del servicio.
- Las simulaciones pueden proporcionar pruebas locales rápidas una vez configuradas.
- Las simulaciones pueden proporcionar un comportamiento sustituto para prácticamente cualquier tipo de objeto, por lo que las estrategias de simulación pueden ofrecer cobertura a una variedad de servicios más amplia que los emuladores.
- Cuando hay nuevas características o comportamientos disponibles, las pruebas simuladas pueden reaccionar más rápidamente. Al utilizar un marco simulado genérico, puede simular nuevas características tan pronto como el AWS SDK actualizado esté disponible.

Las pruebas simuladas tienen las siguientes desventajas:

- Por lo general, las simulaciones requieren un esfuerzo no trivial de preparación y configuración, especialmente cuando se trata de determinar los valores devueltos de diferentes servicios para simular correctamente las respuestas.
- Los desarrolladores deben escribir, configurar y mantener las simulaciones, lo que aumenta sus responsabilidades.
- Es posible que necesite tener acceso a la nube para comprender las API y devolver los valores de los servicios.
- Las simulaciones pueden ser difíciles de mantener. Cuando cambien las firmas de la API en la nube simuladas o evolucionen los esquemas de valores de retorno, debe actualizar las simulaciones. Las simulaciones también requieren actualizaciones si amplía la lógica de la aplicación para realizar llamadas a nuevas API.
- Las pruebas que utilizan simulaciones pueden funcionar en entornos de escritorio pero fallar en la nube. Es posible que los resultados no coincidan con la API actual. La configuración del servicio y las cuotas no se pueden probar.
- Los marcos simulados son limitados a la hora de probar o detectar políticas o limitaciones de cuotas de AWS Identity and Access Management (IAM). Si bien las simulaciones son mejores para simular cuando se produce un error en la autorización o se supera una cuota, las pruebas no pueden determinar qué resultado se producirá realmente en un entorno de producción.

## Pruebas con emulación

Los emuladores suelen ser una aplicación que se ejecuta de forma local y que imita un servicio de producción de AWS.

Los emuladores tienen API que se asemejan a las de la nube y ofrecen valores de retorno similares. También pueden simular los cambios de estado que se inician mediante llamadas a la API. Por ejemplo, puede usar AWS SAM para ejecutar una función con AWS SAM local para emular el servicio Lambda y poder invocar rápidamente una función. Para obtener detalles, consulte [AWS SAM local](#) en la Guía para desarrolladores de AWS Serverless Application Model.

Entre las ventajas de realizar pruebas con emuladores se incluyen las siguientes:

- Los emuladores pueden facilitar las iteraciones y pruebas rápidas del desarrollo local.
- Los emuladores proporcionan un entorno familiar para los desarrolladores acostumbrados a desarrollar código en un entorno local. Por ejemplo, si está familiarizado con el desarrollo de una aplicación de n niveles, es posible que tenga un motor de base de datos y un servidor web,

similares a los que se ejecutan en producción, en su equipo local para proporcionar una capacidad de prueba rápida, local y aislada.

- Los emuladores no requieren ningún cambio en la infraestructura de la nube (como las cuentas en la nube de los desarrolladores), por lo que son fáciles de implementar con los patrones de prueba existentes.

Las pruebas con emuladores tienen las siguientes desventajas:

- Los emuladores pueden ser difíciles de configurar y replicar, especialmente cuando se utilizan en canalizaciones de CI/CD. Esto puede aumentar la carga de trabajo del personal de TI o de los desarrolladores que administran su propio software.
- Las características y las API emuladas suelen quedar un paso por detrás de las actualizaciones del servicio. Esto puede generar errores porque el código probado no coincide con la API real e impide la adopción de nuevas características.
- Los emuladores requieren soporte, actualizaciones, correcciones de errores y mejoras en la paridad de características. Son responsabilidad del autor del emulador, que puede ser una empresa externa.
- Las pruebas que se basan en emuladores pueden ofrecer resultados satisfactorios a nivel local, pero fallan en la nube debido a las políticas de seguridad de producción, a las configuraciones entre servicios o a que se superan las cuotas de Lambda.
- Muchos servicios de AWS no tienen emuladores disponibles. Si confía en la emulación, es posible que no disponga de una opción de prueba satisfactoria para algunas partes de su aplicación.

## Prácticas recomendadas

En las siguientes secciones, se proporcionan recomendaciones para realizar pruebas satisfactorias de aplicaciones sin servidor.

Puede encontrar ejemplos prácticos de pruebas y automatización de pruebas en el [repositorio de ejemplos de pruebas sin servidor](#).

## Priorice las pruebas en la nube

Las pruebas en la nube ofrecen la cobertura de pruebas más confiable, precisa y completa. Al realizar pruebas en el contexto de la nube, se comprobarán de manera exhaustiva no solo la lógica empresarial, sino también las políticas de seguridad, las configuraciones de servicios, las cuotas y las firmas de API y los valores devueltos más actualizados.

## Estructure su código para que sea fácil de probar

Simplifique sus pruebas y funciones de Lambda al separar el código específico de Lambda de su lógica empresarial principal.

El controlador de la función de Lambda debe ser un adaptador compacto que tome los datos de los eventos y que transmita solo los detalles importantes a sus métodos de lógica empresarial. Con esta estrategia, puede empaquetar pruebas exhaustivas en torno a su lógica empresarial sin preocuparse por los detalles específicos de Lambda. Las funciones de AWS Lambda no deberían requerir la configuración de un entorno complejo o una gran cantidad de dependencias para crear e inicializar el componente que se está probando.

En términos generales, debe escribir un controlador que extraiga y valide los datos de los objetos de evento y contexto entrantes y, a continuación, envíe esa entrada a los métodos que ejecuten su lógica empresarial.

## Acelere los ciclos de retroalimentación del desarrollo

Existen herramientas y técnicas para acelerar los ciclos de retroalimentación del desarrollo. Por ejemplo, tanto [AWS SAM Accelerate](#) como el [modo de vigilancia de AWS CDK](#) reducen el tiempo necesario para actualizar los entornos de nube.

Los ejemplos del [repositorio de ejemplos de pruebas sin servidor](#) de GitHub exploran algunas de estas técnicas.

También recomendamos que cree y pruebe los recursos en la nube lo antes posible durante el desarrollo, no solo después de comprobar el control del código fuente. Esta práctica permite una exploración y experimentación más rápidas a la hora de desarrollar soluciones. Además, la automatización de la implementación desde un equipo de desarrollo lo ayuda a descubrir los problemas de configuración de la nube con mayor rapidez y reduce el esfuerzo desperdiciado en las actualizaciones y los procesos de revisión del código.

## Céntrese en las pruebas de integración

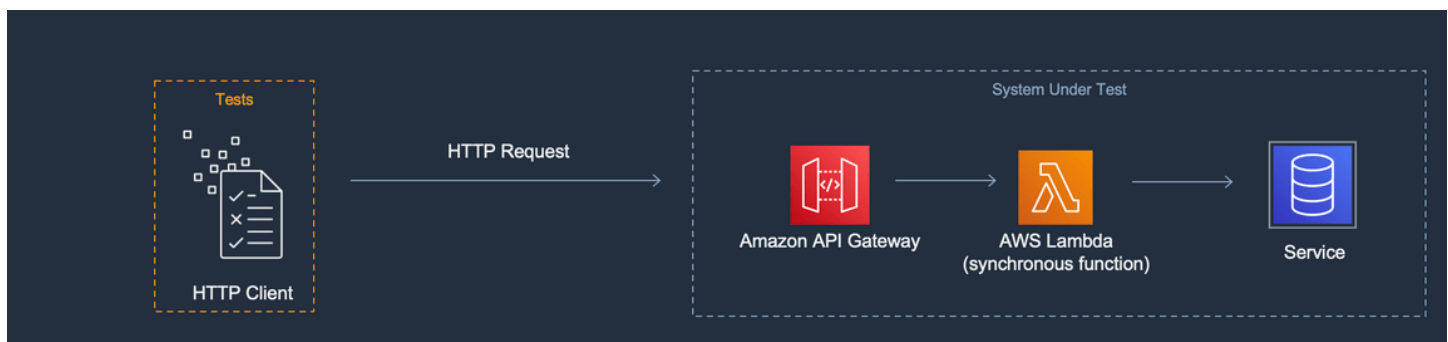
Cuando se crean aplicaciones con Lambda, se recomienda probar los componentes juntos.

Las pruebas que se ejecutan en dos o más componentes arquitectónicos se denominan pruebas de integración. El objetivo de las pruebas de integración es comprender no solo cómo se ejecutará el código en todos los componentes, sino también cómo se comportará el entorno que aloja el

código. Las pruebas integrales son tipos especiales de pruebas de integración que verifican los comportamientos de toda una aplicación.

Para crear pruebas de integración, implemente la aplicación en un entorno de nube. Esto se puede hacer desde un entorno local o mediante una canalización de CI/CD. A continuación, escriba pruebas para ejercitar el sistema bajo prueba (SUT) y validar el comportamiento esperado.

Por ejemplo, el sistema bajo prueba puede ser una aplicación que utilice API Gateway, Lambda y DynamoDB. Una prueba puede realizar una llamada HTTP sintética a un punto de conexión de API Gateway y validar que la respuesta haya incluido la carga esperada. Esta prueba valida que el código de AWS Lambda es correcto y que cada servicio está configurado de forma adecuada para gestionar la solicitud, incluidos los permisos de IAM entre ellos. Además, puede diseñar la prueba para escribir registros de varios tamaños a fin de verificar que sus cuotas de servicio, como el tamaño máximo de registro en DynamoDB, están configuradas correctamente.



## Cree entornos de pruebas aislados

Las pruebas en la nube suelen requerir entornos de desarrollo aislados, de modo que las pruebas, los datos y los eventos no se superpongan.

Un enfoque consiste en proporcionar a cada desarrollador una cuenta de AWS dedicada. Esto evitará conflictos con la nomenclatura de los recursos que pueden ocurrir cuando varios desarrolladores que trabajan en una base de código compartida intentan implementar recursos o invocar una API.

Los procesos de prueba automatizados deben crear recursos con nombres únicos para cada pila. Por ejemplo, puede configurar scripts o archivos de configuración TOML para que los comandos de la CLI de AWS SAM [sam deploy](#) o [sam sync](#) especifiquen automáticamente una pila con un prefijo único.

En algunos casos, los desarrolladores comparten una cuenta de AWS. Esto puede deberse a que hay recursos en su pila que son costosos de operar o de aprovisionar y configurar. Por ejemplo, se

puede compartir una base de datos para facilitar la configuración y la siembra de datos de forma adecuada.

Si los desarrolladores comparten una cuenta, debe establecer límites para identificar la propiedad y eliminar la superposición. Una forma de hacerlo consiste en poner un prefijo a los nombres de las pilas con los ID de usuario de los desarrolladores. Otro enfoque popular es configurar pilas basadas en ramas de código. Con los límites de las ramas, los entornos están aislados, pero los desarrolladores pueden seguir compartiendo recursos, como una base de datos relacional. Este enfoque es una práctica recomendada cuando los desarrolladores trabajan en más de una rama a la vez.

Las pruebas en la nube son valiosas para todas las fases de las pruebas, incluidas las pruebas unitarias, las pruebas de integración y las pruebas integrales. Mantener un aislamiento adecuado es esencial, pero aun así querrá que su entorno de control de calidad se parezca lo más posible a su entorno de producción. Por este motivo, los equipos agregan procesos de control de cambios para los entornos de control de calidad.

En los entornos de preproducción y producción, los límites se suelen establecer a nivel de cuenta para aislar las cargas de trabajo de los problemas de vecinos ruidosos e implementar controles de seguridad con privilegios mínimos a fin de proteger los datos confidenciales. Las cargas de trabajo tienen cuotas. No querrá que sus pruebas consuman las cuotas asignadas a la producción (vecinos ruidosos) ni tengan acceso a los datos de los clientes. Las pruebas de carga son otra actividad que debe aislar de la pila de producción.

En todos los casos, los entornos deben configurarse con alertas y controles para evitar gastos innecesarios. Por ejemplo, puede limitar el tipo, el nivel o el tamaño de los recursos que se pueden crear y configurar alertas por correo electrónico cuando los costos estimados superen un umbral determinado.

## Utilice simulaciones para una lógica empresarial aislada

Los marcos simulados son una herramienta valiosa para escribir pruebas unitarias rápidas. Son especialmente útiles cuando las pruebas cubren una lógica empresarial interna compleja, como cálculos o simulaciones matemáticas o financieras. Busque pruebas unitarias que tengan una gran cantidad de casos de prueba o variaciones de entrada, en las que esas entradas no cambien el patrón ni el contenido de las llamadas a otros servicios en la nube.

El código que se incluye en las pruebas unitarias con simulaciones también debe incluirse en las pruebas en la nube. Esto se recomienda porque un entorno de equipo portátil o de compilación para

desarrolladores puede configurarse de forma diferente a un entorno de producción en la nube. Por ejemplo, las funciones de Lambda pueden utilizar más memoria o tiempo del asignado cuando se ejecutan con ciertos parámetros de entrada. O bien, el código puede incluir variables de entorno que no están configuradas de la misma manera (o no están configuradas en absoluto), y las diferencias pueden provocar que el código se comporte de manera diferente o que falle.

El beneficio de las simulaciones es menor en las pruebas de integración, ya que el nivel de esfuerzo para implementar las simulaciones necesarias aumenta con la cantidad de puntos de conexión. Las pruebas integrales no deben utilizar simulaciones, ya que estas pruebas suelen tratar con estados y lógicas complejas que no se pueden simular fácilmente con marcos simulados.

Por último, evite utilizar servicios en la nube simulados para validar el despliegue correcto de las llamadas de servicio. En su lugar, realice llamadas al servicio de nube en la nube para validar el comportamiento, la configuración y el despliegue funcional.

## Use emuladores con moderación

Los emuladores pueden resultar útiles para algunos casos de uso, por ejemplo, para un equipo de desarrollo con acceso a Internet limitado, poco fiable o lento. Sin embargo, en la mayoría de los casos, opte por utilizar los emuladores con moderación.

Al evitar los emuladores, podrá crear e innovar con las características de servicio más recientes y las API actualizadas. No tendrá que esperar a que los proveedores publiquen sus productos para lograr la paridad de características. Reducirá los gastos iniciales y continuos de compra y configuración en múltiples sistemas de desarrollo y equipos de compilación. Además, evitará el problema de que muchos servicios en la nube simplemente no tienen emuladores disponibles. Una estrategia de pruebas que dependa de la emulación imposibilitará el uso de esos servicios (lo que podría generar soluciones alternativas más costosas) o producirá código y configuraciones que no estén bien probados.

Cuando utilice la emulación para realizar pruebas, deberá seguir realizando pruebas en la nube para verificar la configuración y probar las interacciones con los servicios en la nube que solo se pueden simular o imitar en un entorno emulado.

## Desafíos al probar de forma local

Cuando utilice emuladores y llamadas simuladas para realizar pruebas en el escritorio local, es posible que experimente incoherencias en las pruebas a medida que el código avanza de un entorno

a otro en la canalización de CI/CD. Es posible que las pruebas unitarias para validar la lógica empresarial de la aplicación en el escritorio no prueben con precisión los aspectos críticos de los servicios en la nube.

En los siguientes ejemplos, se muestran casos para tener en cuenta al realizar pruebas locales con simulaciones y emuladores:

## Ejemplo: la función de Lambda crea un bucket de S3

Si la lógica de una función de Lambda depende de la creación de un bucket de S3, una prueba completa debería confirmar que se llamó a Amazon S3 y que el bucket se creó correctamente.

- En una configuración de prueba simulada, puede simular una respuesta correcta y, potencialmente, agregar un caso de prueba para gestionar una respuesta de error.
- En un escenario de prueba de emulación, es posible que se llame a la API `CreateBucket`, pero debe tener en cuenta que la identidad que realiza la llamada local no procederá del servicio Lambda. La identidad de llamada no asumirá un rol de seguridad como lo haría en la nube, por lo que se utilizará en su lugar una autenticación de marcadores de posición, posiblemente con un rol o identidad de usuario más permisiva que será diferente cuando se ejecute en la nube.

Las configuraciones de simulación y emulación comprobarán qué hará la función de Lambda si llama a Amazon S3; sin embargo, esas pruebas no verificarán que la función de Lambda, tal como está configurada, sea capaz de crear correctamente el bucket de Amazon S3. Debe asegurarse de que el rol asignado a la función tenga una política de seguridad adjunta que permita a la función realizar la acción `s3:CreateBucket`. De lo contrario, es probable que la función falle cuando se implemente en un entorno de nube.

## Ejemplo: la función de Lambda procesa mensajes de una cola de Amazon SQS

Si una cola de Amazon SQS es el origen de una función de Lambda, una prueba completa debería comprobar que la función de Lambda se invoca correctamente cuando se coloca un mensaje en una cola.

Las pruebas de emulación y simulación generalmente se configuran para ejecutar el código de la función de Lambda directamente y para simular la integración de Amazon SQS pasando una carga de eventos JSON (o un objeto deserializado) como entrada del controlador de funciones.



Las pruebas locales que simulan la integración de Amazon SQS comprobarán qué hará la función de Lambda cuando Amazon SQS la llame con una carga determinada, pero no verificarán que Amazon SQS invoque correctamente la función de Lambda cuando se implemente en un entorno de nube.

Algunos ejemplos de problemas de configuración que pueden surgir con Amazon SQS y Lambda son los siguientes:

- El tiempo de espera de visibilidad de Amazon SQS es demasiado bajo, lo que provoca varias invocaciones cuando solo se tiene prevista una.
- El rol de ejecución de la función de Lambda no permite leer los mensajes de la cola (a través de `sqs:ReceiveMessage`, `sqs:DeleteMessage` o `sqs:GetQueueAttributes`).
- El evento de ejemplo que se pasa a la función de Lambda supera la cuota de tamaño de los mensajes de Amazon SQS. Por lo tanto, la prueba no es válida porque Amazon SQS nunca podrá enviar un mensaje de ese tamaño.

Como se muestra en estos ejemplos, es probable que las pruebas que cubren la lógica empresarial, pero no las configuraciones entre los servicios en la nube, arrojen resultados poco fiables.

## Preguntas frecuentes

Tengo una función de Lambda que realiza cálculos y devuelve un resultado sin llamar a ningún otro servicio. ¿Realmente necesito probarlo en la nube?

Sí. Las funciones de Lambda tienen parámetros de configuración que pueden cambiar el resultado de la prueba. Todo el código de la función de Lambda depende de la configuración del [tiempo de espera](#) y la [memoria](#), lo que puede provocar un error en la función si esos ajustes no se configuran correctamente. Las políticas de Lambda también permiten el registro de salida estándar en [Amazon CloudWatch](#). Incluso si su código no llama directamente a CloudWatch, se necesitan permisos para habilitar el registro. Este permiso obligatorio no se puede burlar ni emular con precisión.

¿Cómo pueden ayudar las pruebas en la nube a las pruebas unitarias? Si está en la nube y se conecta a otros recursos, ¿no es una prueba de integración?

Definimos las pruebas unitarias como las pruebas que funcionan en componentes arquitectónicos de forma aislada, pero esto no impide que las pruebas incluyan componentes que pueden llamar a otros servicios o utilizar alguna comunicación de red.

Muchas aplicaciones sin servidor tienen componentes arquitectónicos que se pueden probar de forma aislada, incluso en la nube. Un ejemplo es una función de Lambda que toma una entrada,

procesa los datos y envía un mensaje a una cola de Amazon SQS. Una prueba unitaria de esta función probablemente comprobaría si los valores de entrada dan como resultado que ciertos valores estén presentes en el mensaje en cola.

Considere una prueba que se escribe con el patrón Organizar, Actuar, Afirmar:

- Organizar: asigne recursos (una cola para recibir mensajes y la función que se está probando).
- Actuar: llame a la función que se está probando.
- Afirmar: recupere el mensaje enviado por la función y valide la salida.

Un enfoque de prueba simulada implicaría simular la cola con un objeto simulado en proceso y crear una instancia en proceso de la clase o módulo que contenga el código de la función de Lambda. Durante la fase de afirmación, el mensaje en cola se recuperaría del objeto de la simulación.

En un enfoque basado en la nube, la prueba crearía una cola de Amazon SQS para los fines de la prueba e implementaría la función de Lambda con variables de entorno configuradas para utilizar la cola aislada de Amazon SQS como destino de salida. Tras ejecutar la función de Lambda, la prueba recuperaría el mensaje de la cola de Amazon SQS.

La prueba basada en la nube ejecutaría el mismo código, afirmaría el mismo comportamiento y validaría la corrección funcional de la aplicación. Sin embargo, tendría la ventaja adicional de poder validar la configuración de la función de Lambda: el rol de IAM, las políticas de IAM y la configuración de memoria y tiempo de espera de la función.

## Próximos pasos y recursos

Utilice los siguientes recursos para obtener más información y explorar ejemplos prácticos de pruebas.

### Despliegue de ejemplo

El [repositorio de ejemplos de pruebas sin servidor](#) de GitHub contiene ejemplos concretos de pruebas que siguen los patrones y las prácticas recomendadas que se describen en esta guía. El repositorio contiene código de muestra y tutoriales guiados sobre los procesos de simulación, emulación y prueba en la nube descritos en las secciones anteriores. Utilice este repositorio para ponerse al día con la guía de pruebas sin servidor más reciente de AWS.

### Documentación adicional

Visite [Serverless Land](#) para acceder a los blogs, videos y formaciones más recientes sobre tecnologías de AWS sin servidor.

También se recomienda leer las siguientes publicaciones del blog de AWS:

- [Acelerar el desarrollo sin servidores con AWS SAM Accelerate](#) (publicación del blog de AWS)
- [Incrementar la velocidad de desarrollo con CDK Watch](#) (publicación del blog de AWS)
- [Simulacro de integraciones de servicios con AWS Step Functions Local](#) (publicación del blog de AWS)
- [Introducción a la prueba de aplicaciones sin servidor](#) (publicación del blog de AWS)

## Herramientas

- AWS SAM: [Prueba y depuración de aplicaciones sin servidor](#)
- AWS SAM: [Integración con pruebas automatizadas](#)
- Lambda: [Probar las funciones de Lambda en la consola de Lambda](#)

# Invocar Lambda con eventos de otros servicios de AWS

Algunos servicios de AWS pueden invocar directamente las funciones de Lambda mediante desencadenadores. Estos servicios envían eventos a Lambda y la función se invoca inmediatamente cuando se produce el evento especificado. Los desencadenadores son adecuados para eventos discretos y para el procesamiento en tiempo real. Al [crear un desencadenador mediante la consola de Lambda](#), esta interactúa con el servicio de AWS correspondiente para configurar la notificación de eventos en ese servicio. En realidad, el servicio que genera los eventos es el que almacena y administra el desencadenador, no Lambda.

Los eventos son datos estructurados en formato JSON. La estructura JSON varía según el servicio que la genera y el tipo de evento, pero todas contienen los datos que la función necesita para procesar el evento.

Una función puede tener varios desencadenadores. Cada desencadenador actúa como un cliente que invoca su función de manera independiente, y cada evento que Lambda envía a su función tiene datos de un solo desencadenador. Lambda convierte el documento de evento en un objeto y se lo pasa al controlador de la función.

La invocación basada en eventos puede ser [sincrónica](#) o [asincrónica](#), en función del servicio.

- Para la invocación síncrona, el servicio que genera el evento espera la respuesta de la función. Ese servicio define los datos que la función necesita devolver en la respuesta. El servicio controla la estrategia de error, como, por ejemplo, si se debe reintentar en caso de errores.
- Para la invocación asíncrona, Lambda coloca el evento en la cola antes de pasárselo a la función. Cuando Lambda pone en cola el evento, envía inmediatamente una respuesta de operación correcta al servicio que lo generó. Después de que la función procesa el evento, Lambda no devuelve una respuesta al servicio de generación de eventos.

## Creación de un desencadenador

La forma más sencilla de crear un desencadenador es utilizar la consola de Lambda. Al crear un desencadenador mediante la consola, Lambda agrega de forma automática los permisos necesarios a la [política basada en recursos](#) de la función.

Creación de un desencadenador mediante la consola de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.

2. Seleccione la función para la que desee crear un desencadenador.
3. En el panel Información general de la función, elija Agregar desencadenador.
4. Seleccione el servicio de AWS en el que desee invocar su función.
5. Rellene las opciones del panel Configuración del desencadenador y seleccione Agregar. En función del Servicio de AWS que elija para invocar la función, las opciones de configuración del desencadenador serán diferentes.

## Servicios que pueden invocar funciones de Lambda

En la siguiente tabla se enumeran los servicios que pueden invocar funciones de Lambda.

Servicio	Método de invocación
<a href="#">Transmisión gestionada de Amazon para Apache Kafka</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Apache Kafka autoadministrado</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Amazon API Gateway</a>	Invocación síncrona basada en eventos
<a href="#">AWS CloudFormation</a>	Invocación asíncrona basada en eventos
<a href="#">Registros de Amazon CloudWatch</a>	Invocación asíncrona basada en eventos
<a href="#">AWS CodeCommit</a>	Invocación asíncrona basada en eventos
<a href="#">AWS CodePipeline</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Cognito</a>	Invocación síncrona basada en eventos
<a href="#">AWS Config</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Connect</a>	Invocación síncrona basada en eventos
<a href="#">Amazon DynamoDB</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Amazon Elastic File System</a>	Integración especial

Servicio	Método de invocación
<a href="#">Elastic Load Balancing (Equilibrador de carga de aplicación)</a>	Invocación sincrónica basada en eventos
<a href="#">Amazon EventBridge (CloudWatch Events)</a>	Basado en eventos; invocación asíncrona (buses de eventos), invocación asíncrona o sincrónica (canalizaciones y planificaciones)
<a href="#">AWS IoT</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Kinesis</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Amazon Data Firehose</a>	Invocación sincrónica basada en eventos
<a href="#">Amazon Lex</a>	Invocación sincrónica basada en eventos
<a href="#">Amazon MQ</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Amazon Simple Email Service</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Simple Notification Service</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Simple Queue Service</a>	<a href="#">Asignación de orígenes de eventos</a>
<a href="#">Amazon Simple Storage Service (Amazon S3)</a>	Invocación asíncrona basada en eventos
<a href="#">Amazon Simple Storage Service Batch</a>	Invocación sincrónica basada en eventos
<a href="#">Secrets Manager</a>	Integración especial
<a href="#">AWS Step Functions</a>	Invocación sincrónica o asíncrona basada en eventos
<a href="#">Amazon VPC Lattice</a>	Invocación sincrónica basada en eventos
<a href="#">AWS X-Ray</a>	Integración especial



# Uso de Lambda con Apache Kafka autoadministrado

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

Lambda admite [Apache Kafka](#) como un [origen de eventos](#). Apache Kafka es una plataforma de secuencia de eventos de código abierto que admite cargas de trabajo como canalizaciones de datos y análisis de streaming.

Puede utilizar el servicio Kafka administrado por AWS Amazon Managed Streaming for Apache Kafka (Amazon MSK) o un clúster de Kafka autoadministrado. Para conocer detalles sobre el uso de Lambda con Amazon MSK, consulte [Uso de Lambda con Amazon MSK](#).

En este tema se describe cómo utilizar Lambda con un clúster de Kafka autoadministrado. En la terminología de AWS, un clúster autoadministrado incluye clústeres Kafka alojados que no son de AWS. Por ejemplo, puede alojar su clúster de Kafka con un proveedor de servicios en la nube, como [Confluent Cloud](#).

Apache Kafka como origen de eventos funciona de manera similar a utilizar Amazon Simple Queue Service (Amazon SQS) o Amazon Kinesis. Lambda sondea internamente nuevos mensajes del origen de eventos y luego invoca sincrónicamente la función de Lambda objetivo. Lambda lee los mensajes en lotes y los proporciona a su función como carga de eventos. El tamaño máximo del lote es configurable. (El valor predeterminado es 100 mensajes).

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.



Para los orígenes de eventos basados en Kafka, Lambda admite parámetros de control de procesamiento, como los plazos de procesamiento por lotes y el tamaño del lote. Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

Para ver un ejemplo de cómo utilizar Kafka autoadministrado como origen de eventos, consulte [Uso de Apache Kafka autoalojado como origen de eventos para AWS Lambda](#) en el blog de informática de AWS.

## Temas

- [Evento de ejemplo](#)
- [Configuración de Apache Kafka autoadministrado como origen de eventos para Lambda](#)
- [Procesamiento de mensajes de Apache Kafka autoadministrado con Lambda](#)
- [Filtrado de eventos con un origen de eventos de Apache Kafka autoadministrado](#)
- [Captura de lotes descartados para un origen de eventos de Apache Kafka autoadministrado](#)
- [Solución de errores de asignación de orígenes de eventos de Apache Kafka autoadministrado](#)

## Evento de ejemplo

Lambda envía el lote de mensajes en el parámetro de evento cuando invoca su función de Lambda. La carga de eventos contiene una matriz de mensajes. Cada elemento de la matriz contiene detalles del tema Kafka y el identificador de partición Kafka, junto con una marca de tiempo y un mensaje codificado en base64.

```
{
  "eventSource": "SelfManagedKafka",
  "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092",
  "records": {
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXYZ1234==",
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg=="
      }
    ]
  }
}
```

```
    "headers": [
      {
        "headerKey": [
          104,
          101,
          97,
          100,
          101,
          114,
          86,
          97,
          108,
          117,
          101
        ]
      }
    ]
  }
]
```

## Configuración de Apache Kafka autoadministrado como origen de eventos para Lambda

Antes de crear una asignación de orígenes de eventos para el clúster de Apache Kafka autoadministrado, debe asegurarse de que tanto el clúster como la VPC en la que reside estén correctamente configurados. También debe asegurarse de que el [rol de ejecución](#) de la función de Lambda tenga los permisos de IAM necesarios.

Siga las instrucciones en las siguientes secciones para configurar el clúster Apache Kafka autoadministrado y la función de Lambda. Para obtener información sobre cómo crear la asignación de orígenes de eventos, consulte [the section called “Agregar un clúster de Kafka como origen de eventos”](#).

### Temas

- [Autenticación de clústeres de Kafka](#)
- [Configuración de red](#)
- [Permisos de función de Lambda y acceso a la API](#)

## Autenticación de clústeres de Kafka

Lambda admite varios métodos para autenticarse con su clúster de Apache Kafka autoadministrado. Asegúrese de configurar el clúster de Kafka para que utilice uno de estos métodos de autenticación admitidos: Para obtener más información acerca de la seguridad de Kafka, consulte la sección [Security](#) (Seguridad) de la documentación de Kafka.

### Acceso mediante VPC

Si solo los usuarios de Kafka de su VPC acceden a sus agentes de Kafka, debe configurar el origen de eventos de Kafka para el acceso a Amazon Virtual Private Cloud (Amazon VPC).

### Autenticación SASL/SCRAM

Lambda es compatible con la autenticación simple y la autenticación de capa de seguridad/ mecanismo de autenticación de respuesta por desafío saltado (SASL/SCRAM) con cifrado de seguridad de la capa de transporte (TLS) (SASL\_SSL). Lambda envía las credenciales cifradas para autenticarse con el clúster. Lambda no es compatible con SASL/SCRAM con texto simple (SASL\_PLAINTEXT). Para obtener más información acerca de la autenticación SASL/SCRAM, consulte [RFC 5802](#).

Lambda también admite la autenticación SASL/PLAIN. Dado que este mecanismo utiliza credenciales en texto claro, la conexión con el servidor debe utilizar cifrado TLS para garantizar la protección de las credenciales.

Para la autenticación SASL, almacene las credenciales de inicio de sesión como secreto en AWS Secrets Manager. Para obtener más información sobre cómo utilizar Secrets Manager, consulte [Tutorial: Cree y recupere un secreto](#) en la Guía del usuario de AWS Secrets Manager.

#### Important

Para utilizar Secrets Manager para la autenticación, los secretos deben almacenarse en la misma región de AWS que la función de Lambda.

### Autenticación TLS mutua

TLS mutua (mTLS) proporciona autenticación bidireccional entre el cliente y el servidor. El cliente envía un certificado al servidor para que el servidor verifique el cliente, mientras que el servidor envía un certificado al cliente para que el cliente verifique el servidor.

En Apache Kafka autoadministrado, Lambda actúa como cliente. Puede configurar un certificado de cliente (como secreto en Secrets Manager) para autenticar a Lambda con los agentes de Kafka. El certificado de cliente debe estar firmado por una entidad de certificación en el almacén de confianza del servidor.

El clúster de Kafka envía un certificado de servidor a Lambda para autenticar a los agentes de Kafka con Lambda. El certificado de servidor puede ser un certificado de entidad de certificación pública o un certificado autofirmado o de entidad de certificación privada. El certificado de entidad de certificación pública debe estar firmado por una entidad de certificación que esté en el almacén de confianza de Lambda. Para un certificado autofirmado o de entidad de certificación privada, configure el certificado de entidad de certificación raíz del servidor (como secreto en Secrets Manager). Lambda utiliza el certificado raíz para verificar los agentes de Kafka.

Para obtener más información acerca de mTLS, consulte [Introducing mutual TLS authentication for Amazon MSK as an event source](#) (Presentación de la autenticación de TLS mutua para Amazon MSK como origen de eventos).

### Configuración del secreto de certificado de cliente

El secreto CLIENT\_CERTIFICATE\_TLS\_AUTH requiere un campo de certificado y un campo de clave privada. Para una clave privada cifrada, el secreto requiere una contraseña de clave privada. El certificado y la clave privada deben estar en formato PEM.

#### Note

Lambda admite los algoritmos de cifrado de claves privadas [PBES1](#) (pero no PBES2).

El campo de certificado debe contener una lista de certificados y debe comenzar por el certificado de cliente, seguido de cualquier certificado intermedio, y finalizar con el certificado raíz. Cada certificado debe comenzar en una nueva línea con la siguiente estructura:

```
-----BEGIN CERTIFICATE-----  
    <certificate contents>  
-----END CERTIFICATE-----
```

Secrets Manager admite secretos de hasta 65 536 bytes, que supone suficiente espacio para cadenas de certificados largas.

El formato de la clave privada debe ser [PKCS #8](#), con la siguiente estructura:

```
-----BEGIN PRIVATE KEY-----
    <private key contents>
-----END PRIVATE KEY-----
```

Para una clave privada cifrada, utilice la siguiente estructura:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
    <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

El siguiente ejemplo muestra el contenido de un secreto para la autenticación de mTLS mediante una clave privada cifrada. Para una clave privada cifrada, incluya la contraseña de la clave privada en el secreto.

```
{
  "privateKeyPassword": "testpassword",
  "certificate": "-----BEGIN CERTIFICATE-----
MIIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2K1mII36dg4IMzNjAFEBZiCRoPim040s1cRqtFHXoa10QQbI1xk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdJNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBB
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
  "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDANBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzg09IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

Configuración del secreto de certificado de entidad de certificación raíz del servidor

Cree este secreto si sus agentes de Kafka utilizan cifrado TLS con certificados firmados por una entidad de certificación privada. Puede utilizar el cifrado TLS para autenticación VPC, SASL/SCRAM, SASL/PLAIN o mTLS.

El secreto de certificado de entidad de certificación raíz del servidor requiere un campo que contenga el certificado de entidad de certificación raíz del agente de Kafka en formato PEM. La estructura del secreto se muestra en el ejemplo siguiente.

```
{"certificate": "-----BEGIN CERTIFICATE-----  
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMCVVMx  
EDA0BgNVBAgTB0FyaXpvbmExEzARBgNVBAcTC1Njb3R0c2RhbGUxJTAjBgNVBAoT  
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4x0zA5BgNVBAMTM1N0YXJmaWVs  
ZCBTZXJ2aWN1cyBSb290IEN1cnRpZm1jYXR1IEF1dG...  
-----END CERTIFICATE-----"  
}
```

## Configuración de red

Para que Lambda utilice su clúster de Kafka como origen de eventos, necesita acceder a la Amazon VPC en la que reside su clúster. Le recomendamos que implemente [puntos de conexión de VPC](#) AWS PrivateLink para que Lambda acceda a su VPC. Implemente puntos de conexión para Lambda y AWS Security Token Service (AWS STS). Si el agente utiliza autenticación, implemente también un punto de conexión de VPC para Secrets Manager. Si configuró un [destino en caso de error](#), implemente también un punto de conexión de VPC para el servicio de destino.

De manera alternativa, asegúrese de que la VPC asociada a su clúster de Kafka incluya una gateway NAT por subred pública. Para obtener más información, consulte [the section called “Acceso a Internet para funciones de VPC”](#).

Si usa puntos de enlace de VPC, también debe configurarlos para [habilitar los nombres DNS privados](#).

Al crear una asignación de orígenes de eventos para un clúster autoadministrado de Apache Kafka, Lambda comprueba si las interfaces de red elásticas (ENI) ya están presentes en las subredes y los grupos de seguridad de su VPC del clúster. Si Lambda encuentra ENI existentes, intenta reutilizarlos. De lo contrario, Lambda crea nuevos ENI para conectarse al origen de eventos e invocar la función.

### Note

Las funciones de Lambda siempre se ejecutan dentro de VPC propiedad del servicio de Lambda. El servicio mantiene estas VPC de forma automática, y no son visibles para los clientes. También puede conectar su función a una Amazon VPC. En cualquier caso, la configuración de VPC de la función no afecta la asignación de orígenes de eventos. Solo la

configuración de la VPC del origen de eventos determina cómo Lambda se conecta al origen de eventos.

Para obtener más información sobre la configuración de la red, consulte [Configuración de AWS Lambda con un clúster Apache Kafka dentro de una VPC](#) en el Blog de informática de AWS.

## Reglas de los grupos de seguridad de VPC

Configure los grupos de seguridad de Amazon VPC que contienen su clúster con las siguientes reglas (como mínimo):

- Reglas de entrada: permiten todo el tráfico en el puerto del agente de Kafka para los grupos de seguridad especificados para el origen de eventos. De forma predeterminada, Kafka utiliza el puerto 9092.
- Reglas de salida: permiten todo el tráfico en el puerto 443 para todos los destinos. Permiten todo el tráfico en el puerto del agente de Kafka para los grupos de seguridad especificados para el origen de eventos. De forma predeterminada, Kafka utiliza el puerto 9092.
- Si utiliza puntos de conexión de VPC en lugar de una puerta de enlace NAT, los grupos de seguridad asociados a los puntos de conexión de VPC deben permitir todo el tráfico entrante en el puerto 443 desde los grupos de seguridad del origen de eventos.

## Uso de los puntos de enlace de la VPC

Cuando utiliza puntos de conexión de VPC, las llamadas a la API para invocar su función se enrutan a través de estos puntos de conexión mediante los ENI. La entidad principal del servicio de Lambda debe llamar a `sts:AssumeRole` y `lambda:InvokeFunction` para cualquier rol y función que utilicen esos ENI.

De forma predeterminada, los puntos de conexión de VPC tienen políticas de IAM abiertas. La práctica recomendada es restringir estas políticas para permitir que solo entidades principales específicas realicen las acciones necesarias mediante ese punto de conexión. Para garantizar que la asignación de orígenes de eventos pueda invocar la función de Lambda, la política de puntos de conexión de VPC debe permitir que el principio del servicio de Lambda llame a `sts:AssumeRole` y `lambda:InvokeFunction`. Restringir las políticas de puntos de conexión de VPC para permitir únicamente las llamadas a la API que se originen en su organización impide que la asignación de orígenes de eventos funcione correctamente.

En el siguiente ejemplo de políticas de puntos de conexión de VPC, se muestra cómo conceder el acceso necesario a las entidades principales del servicio de Lambda para AWS STS y los puntos de conexión de Lambda.

#### Example Política del punto de conexión de VPC: punto de conexión de AWS STS

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

#### Example Política del punto de conexión de VPC: punto de conexión de Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

Si su agente de Kafka utiliza la autenticación, también puede restringir la política de puntos de conexión de VPC para el punto de conexión de Secrets Manager. Para llamar a la API de Secrets Manager, Lambda usa su rol de función, no la entidad principal de servicio de Lambda. En el siguiente ejemplo, se muestra una política del punto de conexión de Secrets Manager.



## Example Política de punto de conexión de VPC: punto de conexión de Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "customer_function_execution_role_arn"
        ]
      },
      "Resource": "customer_secret_arn"
    }
  ]
}
```

Si tiene configurado un destino en caso de error, Lambda también utiliza el rol de su función para llamar a `s3:PutObject`, `sns:Publish` o `sqs:sendMessage` mediante los ENI administrados por Lambda.

## Permisos de función de Lambda y acceso a la API

Además de acceder al clúster de Kafka autoadministrado, la función de Lambda necesita permisos para llevar a cabo varias acciones de la API. Los permisos se agregan al [rol de ejecución](#) de la función. Si los usuarios tienen que acceder a cualquier acción de la API, agregue los permisos necesarios a la política de identidad para el usuario o rol de AWS Identity and Access Management (IAM).

### Permisos de función de Lambda necesarios

Para crear y almacenar registros en un grupo de registros en Registros de Amazon CloudWatch, la función de Lambda debe tener los siguientes permisos en su rol de ejecución:

- [logs:CreateLogGroup](#)
- [logs:CreateLogStream](#)
- [logs:PutLogEvents](#)

### Permisos de función de Lambda opcionales

Es posible que la función de Lambda también necesite permisos para:

- Describir el secreto de Secrets Manager.
- Acceder a su clave administrada por el cliente de AWS Key Management Service (AWS KMS).
- Acceder a su Amazon VPC.
- Envíe los registros de las invocaciones fallidas a un destino.

## Secrets Manager y permisos de AWS KMS

En función del tipo de control de acceso que configure para los agentes de Kafka, es posible que la función de Lambda necesite permiso para acceder a su secreto de Secrets Manager o para descifrar su clave administrada por el cliente de AWS KMS. Para acceder a estos recursos, el rol de ejecución de la función debe tener los siguientes permisos:

- [secretsmanager:GetSecretValue](#)
- [kms:Decrypt](#)

## Permisos de VPC

Si solo los usuarios de una VPC pueden acceder a su clúster de Apache Kafka autoadministrado, su función de Lambda debe tener permiso para acceder a sus recursos de Amazon VPC. Estos recursos incluyen su VPC, subredes, grupos de seguridad e interfaces de red. Para acceder a estos recursos, el rol de ejecución de la función debe tener los siguientes permisos:

- [ec2:CreateNetworkInterface](#)
- [ec2:DescribeNetworkInterfaces](#)
- [ec2:DescribeVpcs](#)
- [ec2:DeleteNetworkInterface](#)
- [ec2:DescribeSubnets](#)
- [ec2:DescribeSecurityGroups](#)

## Adición de permisos a su rol de ejecución

Para tener acceso a otros servicios de AWS que su clúster Apache Kafka autoadministrado utiliza, Lambda utiliza las políticas de permisos que defina en el [rol de ejecución](#) de la función de Lambda.

De forma predeterminada, Lambda no está permitido realizar las acciones necesarias u opcionales para un clúster Apache Kafka autoadministrado. Debe crear y definir estas acciones en una [política](#)

[de confianza de IAM](#) y, a continuación, adjuntarla a su rol de ejecución. En este ejemplo se muestra cómo puede crear una política que permita a Lambda tener acceso a los recursos de Amazon VPC.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

Para obtener información sobre cómo crear un documento de política de JSON en la consola de IAM, consulte [Creación de políticas en la pestaña JSON](#) en la Guía del usuario de IAM.

## Concesión de acceso a los usuarios con una política de IAM

De forma predeterminada, los usuarios y roles no tienen permiso para llevar a cabo [operaciones de API de origen de eventos](#). Para conceder acceso a los usuarios de su organización o cuenta, cree o actualice la política basada en identidades. Para obtener más información, consulte [Control del acceso a los recursos de AWS mediante políticas](#) en la Guía del usuario de IAM.

## Procesamiento de mensajes de Apache Kafka autoadministrado con Lambda

### Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

## Temas

- [Agregar un clúster de Kafka como origen de eventos](#)
- [Parámetros de configuración de Apache Kafka autoadministrado](#)
- [Utilizar un clúster de Kafka como origen de eventos](#)
- [Posiciones iniciales de flujos y sondeo](#)
- [Escalado automático del origen de eventos Kafka](#)
- [Métricas de Amazon CloudWatch](#)

## Agregar un clúster de Kafka como origen de eventos

Para crear una [asignación de orígenes de eventos](#), agregue su clúster de Kafka como un [desencadenador](#) de una función de Lambda a través de la consola de Lambda, un [AWS SDK](#), o el [AWS Command Line Interface \(AWS CLI\)](#).

En esta sección se describe cómo crear una asignación de orígenes de eventos mediante la consola de Lambda y la AWS CLI.

### Requisitos previos

- Un clúster de Apache Kafka autoadministrado. Lambda es compatible con Apache Kafka 0.10.1.0 y versiones posteriores.
- Un [rol de ejecución](#) con permiso para acceder a los recursos de AWS que utiliza el clúster de Kafka autoadministrado.

### ID del grupo de consumidores personalizable

Al configurar Kafka como origen de eventos, puede especificar un ID de grupo de consumidores. Este ID de grupo de consumidores es un identificador existente para el grupo de consumidores de Kafka al que desea que se una la función de Lambda. Puede utilizar esta característica para migrar sin problemas cualquier configuración de procesamiento de registro de Kafka en curso de otros consumidores a Lambda.

Si especifica un ID de grupo de consumidores y hay otros sondeadores activos dentro de ese grupo de consumidores, Kafka distribuirá los mensajes entre todos los consumidores. En otras palabras, Lambda no recibe todos los mensajes del tema Kafka. Si desea que Lambda gestione todos los mensajes del tema, desactive los demás sondeadores de ese grupo de consumidores.

Además, si especifica un ID de grupo de consumidores y Kafka encuentra un grupo de consumidores existente válido con el mismo ID, Lambda ignora el parámetro `StartingPosition` para la asignación de orígenes de eventos. En cambio, Lambda comienza a procesar los registros de acuerdo con la compensación comprometida del grupo de consumidores. Si especifica un ID de grupo de consumidores y Kafka no puede encontrar un grupo de consumidores existente, Lambda configura el origen de eventos con el `StartingPosition` especificado.

El ID del grupo de consumidores que especifique debe ser único entre todos los orígenes de eventos de Kafka. Tras crear una asignación de origen de eventos de Kafka con el ID de grupo de consumidores especificado, no puede actualizar este valor.

### Agregar un clúster de Kafka autoadministrado (consola)

Siga estos pasos para agregar su clúster Apache Kafka autoadministrado y un tema Kafka como desencadenador de su función de Lambda.

Para agregar un desencadenador de Apache Kafka a su función de Lambda (consola)

1. Abra la página de [Functions \(Funciones\)](#) en la consola de Lambda.
2. Elija el nombre de su función de Lambda.
3. En Descripción general de la función, elija Agregar desencadenador.
4. En Configuración del desencadenador, haga lo siguiente:
  - a. Elija el tipo de desencadenador Apache Kafka.
  - b. Para los servidores de Bootstrap, ingrese la dirección de host y par de puertos de un broker de Kafka en su clúster y, a continuación, elija Add (Agregar). Repita para cada broker de Kafka en el clúster.
  - c. Para el nombre del tema, escriba el nombre del tema Kafka utilizado para almacenar registros en el clúster.
  - d. (Opcional) Para Tamaño del lote, introduzca el número máximo de registros que se recibirán en un solo lote.
  - e. Para el periodo de lotes, ingrese la cantidad máxima de segundos que Lambda emplea a fin de recopilar registros antes de invocar la función.
  - f. (Opcional) Para el ID del grupo de consumidores, ingrese el ID de un grupo de consumidores de Kafka al que unirse.

- g. (Opcional) En Posición inicial, elija Última para empezar a leer el flujo desde el registro más reciente, Horizonte de supresión para comenzar por el registro más antiguo disponible o En la marca de tiempo para especificar una marca de tiempo desde la cual comenzar a leer.
- h. (Opcional) Para VPC, elija Amazon VPC para su clúster de Kafka. A continuación, elija VPC subnets (Subredes de VPC) y VPC security groups (Grupos de seguridad de VPC).

Esta configuración es obligatoria si solo los usuarios de la VPC acceden a los agentes.

- i. (Opcional) Para Authentication (Autenticación), elija Add (Agregar) y, a continuación, haga lo siguiente:
  - i. Elija el protocolo de acceso o autenticación de los agentes de Kafka en su clúster.
    - Si su agente de Kafka utiliza autenticación SASL/PLAIN, elija BASIC\_AUTH.
    - Si su agente utiliza autenticación de SASL/SCRAM, elija uno de los protocolos de SASL\_SCRAM.
    - Si configura la autenticación de mTLS, elija el protocolo CLIENT\_CERTIFICATE\_TLS\_AUTH.
  - ii. Para la autenticación de SASL/SCRAM o mTLS, elija la clave secreta de Secrets Manager que contiene las credenciales del clúster de Kafka.
- j. (Opcional) Para Encryption (Cifrado), elija el secreto de Secrets Manager que contiene el certificado de entidad de certificación raíz que los agentes de Kafka utilizan para el cifrado con TLS, si los agentes de Kafka utilizan certificados firmados por una entidad de certificación privada.

Esta configuración se aplica al cifrado con TLS para SASL/SCRAM o SASL/PLAIN y a la autenticación con mTLS.

- k. Para crear el desencadenador en un estado deshabilitado para la prueba (recomendado), desactive Activar desencadenador. O bien, para habilitar el desencadenador de inmediato, seleccione Activar desencadenador.
5. Para crear el desencadenador, elija Add (Añadir).

### Agregar un clúster Kafka autoadministrado (AWS CLI)

Utilice los siguientes AWS CLI comandos de ejemplo para crear y ver un desencadenador Apache Kafka autoadministrado para su función de Lambda.

## Uso de SASL/SCRAM

Si los usuarios de Kafka acceden a los agentes de Kafka a través de Internet, especifique el secreto de Secrets Manager que creó para la autenticación con SASL/SCRAM. El siguiente ejemplo utiliza el comando de AWS CLI [create-event-source-mapping](#) para asignar una función de Lambda llamada `my-kafka-function` a un tema de Kafka llamado `AWSKafkaTopic`.

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-east-1:111122223333:secret:MyBrokerSecretName \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

## Uso de una VPC

Si solo los usuarios de Kafka de su VPC acceden a sus agentes de Kafka, debe especificar su VPC, subredes y grupo de seguridad de VPC. El siguiente ejemplo utiliza el comando [create-event-source-mapping](#) AWS CLI para mapear una función de Lambda llamada `my-kafka-function` a un tema de Kafka llamado `AWSKafkaTopic`.

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration '[{"Type": "VPC_SUBNET", "URI": "subnet:subnet-0011001100"}, {"Type": "VPC_SUBNET", "URI": "subnet:subnet-0022002200"}, {"Type": "VPC_SECURITY_GROUP", "URI": "security_group:sg-0123456789"}]' \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

## Visualización del estado mediante la AWS CLI

El siguiente ejemplo utiliza el comando [get-event-source-mapping](#) AWS CLI para describir el estado de la asignación de orígenes de eventos que ha creado.

```
aws lambda get-event-source-mapping
  --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

## Parámetros de configuración de Apache Kafka autoadministrado

Todos los tipos de fuente de eventos Lambda comparten las mismas operaciones

[CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Apache Kafka.

Parámetro	Obligatoria	Predeterminado	Notas
BatchSize	N	100	Máximo: 10 000
Habilitado	N	Habilitado	Ninguno
FunctionName	Y	N/A	Ninguno
FilterCriteria	N	N/A	<a href="#">Controle qué eventos envía Lambda a la función</a>
MaximumBatchingWindowInSeconds	N	500 ms	<a href="#">Comportamiento de procesamiento por lotes</a>
SelfManagedEventSource	Y	N/A	Lista de agentes de Kafka. Solo se puede establecer en Crear
SelfManagedKafkaEventSourceConfig	N	Contiene el campo ConsumerGroupId, que se establece de forma predeterminada en un valor único.	Solo se puede establecer en Crear
SourceAccessConfigurations	N	Sin credenciales	Información de VPC o credenciales de autenticación para el clúster



Parámetro	Obligatoria	Predeterminado	Notas
			Para SASL_PLAIN, establezca en BASIC_AUTH
StartingPosition	Y	N/A	AT_TIMESTAMP, TRIM_HORIZON o LATEST  Solo se puede establecer en Crear
StartingPositionTimestamp	N	N/A	Obligatorio si StartingPosition se establece en AT_TIMESTAMP
Temas	Y	N/A	Nombre del tema  Solo se puede establecer en Crear

## Utilizar un clúster de Kafka como origen de eventos

Cuando agrega su clúster de Apache Kafka o Amazon MSK como desencadenador para su función de Lambda, el clúster se utiliza como [origen de eventos](#).

Lambda lee los datos de eventos de los temas de Kafka que especifique como Topics en una solicitud de [CreateEventSourceMapping](#), en función de la StartingPosition que especifique. Después de un procesamiento exitoso, su tema de Kafka se compromete a su clúster de Kafka.

Si especifica la StartingPosition como LATEST, Lambda comienza a leer el último mensaje de cada partición que pertenece al tema. Debido a que puede haber algún retraso después de la configuración del desencadenador antes de que Lambda comience a leer los mensajes, Lambda no lee ningún mensaje producido durante este plazo.

Lambda procesa registros de una o más particiones de temas de Kafka que especifique y envía una carga JSON a su función. Cuando hay más registros disponibles, Lambda continúa

procesando registros en lotes, en función del valor `BatchSize` que especifique en una solicitud de [CreateEventSourceMapping](#), hasta que la función se ponga al día con el tema.

Si su función devuelve un error para cualquiera de los mensajes de un lote, Lambda reintenta todo el lote de mensajes hasta que el procesamiento sea correcto o los mensajes caduquen. Puede enviar los registros con error en todos los reintentos a un [destino en caso de error](#) para su posterior procesamiento.

#### Note

Si bien las funciones de Lambda suelen tener un límite de tiempo de espera máximo de 15 minutos, las asignaciones de orígenes de eventos para Amazon MSK, Apache Kafka autoadministrado, Amazon DocumentDB y Amazon MQ para ActiveMQ y RabbitMQ solo admiten funciones con límites de tiempo de espera máximos de 14 minutos. Esta restricción garantiza que la asignación de orígenes de eventos pueda gestionar correctamente los errores y reintentos de las funciones.

## Posiciones iniciales de flujos y sondeo

Tenga en cuenta que el sondeo de flujos durante la creación y las actualizaciones de la asignación de orígenes de eventos es, en última instancia, coherente.

- Durante la creación de la asignación de orígenes de eventos, es posible que se demore varios minutos en iniciar el sondeo de los eventos del flujo.
- Durante las actualizaciones de la asignación de orígenes de eventos, es posible que se demore varios minutos en detener y reiniciar el sondeo de los eventos del flujo.

Este comportamiento significa que, si especifica `LATEST` como posición inicial del flujo, la asignación de orígenes de eventos podría omitir eventos durante la creación o las actualizaciones. Para garantizar que no se pierda ningún evento, especifique la posición inicial del flujo como `TRIM_HORIZON` o `AT_TIMESTAMP`.

## Escalado automático del origen de eventos Kafka

Al crear inicialmente un [origen de eventos](#) de Apache Kafka, Lambda asigna un consumidor para procesar todas las particiones en el tema de Kafka. Cada consumidor tiene varios procesadores que se ejecutan en paralelo para gestionar el aumento de las cargas de trabajo. Además, Lambda escala

o reduce verticalmente de manera automática el número de consumidores, en función de la carga de trabajo. Para conservar el orden de mensajes en cada partición, el número máximo de consumidores es un consumidor por partición en el tema.

En intervalos de un minuto, Lambda evalúa el retraso de compensación del consumidor de todas las particiones del tema. Si el retraso es demasiado alto, la partición recibe mensajes más rápido de lo que Lambda puede procesarlos. Si es necesario, Lambda agrega o elimina a los consumidores del tema. El proceso de escalado para agregar o eliminar consumidores se produce dentro de los tres minutos posteriores a la evaluación.

Si su función Lambda objetivo está sobrecargada, Lambda reduce el número de consumidores. Esta acción reduce la carga de trabajo de la función al reducir el número de mensajes que los consumidores pueden recuperar y enviar a la función.

Para monitorear el rendimiento de su tema de Kafka, puede ver las métricas de consumo de Apache Kafka, como `consumer_lag` y `consumer_offset`. Para comprobar cuántas invocaciones de función se producen en paralelo, también puede supervisar las [métricas de simultaneidad](#) para su función.

## Métricas de Amazon CloudWatch

Lambda emite la métrica `OffsetLag` mientras la función procesa los registros. El valor de esta métrica es la diferencia de compensación entre el último registro escrito en el tema de origen de eventos de Kafka y el último registro que procesó el grupo de consumidores de su función. Puede utilizar `OffsetLag` para estimar la latencia entre el momento en el que se agrega un registro y el momento en el que el grupo lo procesa.

Una tendencia ascendente en `OffsetLag` puede indicar problemas con los sondeadores en el grupo de consumidores de su función. Para obtener más información, consulte [Ver las métricas de funciones de Lambda](#).

## Filtrado de eventos con un origen de eventos de Apache Kafka autoadministrado

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo se filtran los eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para orígenes de eventos de Apache Kafka autoadministrado.

## Temas

- [Conceptos básicos del filtrado de eventos de Apache Kafka autoadministrado](#)

### Conceptos básicos del filtrado de eventos de Apache Kafka autoadministrado

Supongamos que un productor escribe mensajes a un tema de su clúster de Apache Kafka autoadministrado, ya sea en formato JSON válido o como cadenas simples. Un registro de ejemplo tendría el siguiente aspecto, con el mensaje convertido en una cadena codificada en Base64 en el campo `value`.

```
{
  "mytopic-0": [
    {
      "topic": "mytopic",
      "partition": 0,
      "offset": 15,
      "timestamp": 1545084650987,
      "timestampType": "CREATE_TIME",
      "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
      "headers": []
    }
  ]
}
```

Supongamos que su productor de Apache Kafka escribe mensajes a su tema en el siguiente formato JSON.

```
{
  "device_ID": "AB1234",
  "session": {
    "start_time": "yyyy-mm-ddThh:mm:ss",
    "duration": 162
  }
}
```

Puede utilizar la clave `value` para filtrar registros. Supongamos que desea filtrar solo los registros en los que `device_ID` comience con las letras AB. El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
```

```

    {
      "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
    }
  ]
}

```

Para mayor claridad, este es el valor del Pattern del filtro ampliado en JSON no cifrado.

```

{
  "value": {
    "device_ID": [ { "prefix": "AB" } ]
  }
}

```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }
```

### AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```

aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/  
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :  
[ { \"prefix\": \"AB\" } ] } }"]}'

```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
```

```
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
--filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"]}]'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }'
```

Con Apache Kafka autoadministrado, también puede filtrar registros en los que el mensaje sea una cadena simple. Supongamos que desea ignorar los mensajes en los que la cadena es “error”. El objeto `FilterCriteria` tendría el siguiente aspecto.

```
{
  "Filters": [
    {
      "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "value": [
    {
      "anything-but": [ "error" ]
    }
  ]
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
[ \"error\" ] } ] }"]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
[ \"error\" ] } ] }"]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

Los mensajes de Apache Kafka autoadministrado deben ser cadenas codificadas en UTF-8, cadenas simples o en formato JSON. Esto se debe a que Lambda decodifica las matrices de bytes de Kafka

en UTF-8 antes de aplicar los criterios de filtrado. Si los mensajes utilizan otra codificación, como UTF-16 o ASCII, o el formato del mensaje no coincide con el formato de `FilterCriteria`, Lambda solo procesa los filtros de metadatos. En la siguiente tabla se resume el comportamiento específico:

Formato del mensaje entrante	Formato del patrón de filtro para las propiedades del mensaje	Acción resultante
Cadena sin formato	Cadena sin formato	Lambda filtra en función de los criterios de filtro.
Cadena sin formato	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
Cadena sin formato	JSON válido	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Cadena sin formato	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.
Cadena no codificada con UTF-8	JSON, cadena sin formato o sin patrón	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.



## Captura de lotes descartados para un origen de eventos de Apache Kafka autoadministrado

Para retener los registros de las invocaciones de asignación de orígenes de eventos fallidos, agregue un destino a la asignación de orígenes de eventos de su función. Cada registro enviado al destino es un documento JSON con metadatos sobre la invocación fallida. Puede configurar cualquier tema de Amazon SNS, cola de Amazon SQS o bucket de S3 como destino. Su rol de ejecución debe tener permisos para el destino:

- Para destinos de SQS: [sqs:SendMessage](#)
- Para destinos de SNS: [sns:Publish](#)
- Para destinos de bucket de S3: [s3:PutObject](#) y [s3:ListBuckets](#)

Además, si configuró una clave de KMS en su destino, Lambda necesita los siguientes permisos según el tipo de destino:

- Si tiene activado el cifrado con su propia clave de KMS para un destino de S3, necesitará [kms:GenerateDataKey](#). Si la clave de KMS y el destino del bucket de S3 están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:GenerateDataKey`.
- Si tiene activado el cifrado con su propia clave de KMS para un destino de SQS, necesitará [kms:Decrypt](#) y [kms:GenerateDataKey](#). Si la clave de KMS y el destino de la cola de SQS están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:Decrypt`, `kms:GenerateDataKey`, [kms:DescribeKey](#) y [kms:ReEncrypt](#).
- Si tiene activado el cifrado con su propia clave de KMS para un destino de SNS, necesitará [kms:Decrypt](#) y [kms:GenerateDataKey](#). Si la clave de KMS y el destino del tema de SNS están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:Decrypt`, `kms:GenerateDataKey`, [kms:DescribeKey](#) y [kms:ReEncrypt](#).

## Configuración de destinos en caso de error para la asignación de orígenes de eventos de Apache Kafka autoadministrado

Para configurar un destino en caso de error mediante la consola, siga estos pasos:

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En Descripción general de la función, elija Agregar destino.
4. En Origen, elija Invocación de asignación de orígenes de eventos.
5. Para la Asignación de orígenes de eventos, elija un origen de eventos que esté configurado para esta función.
6. En Condición, seleccione En caso de error. Para las invocaciones de asignación de orígenes de eventos, esta es la única condición aceptada.
7. En Tipo de destino, elija el tipo de destino al que Lambda envía los registros de invocación.
8. En Destino, elija un recurso.
9. Seleccione Guardar.

También puede configurar un destino en caso de error mediante la AWS CLI. Por ejemplo, el siguiente comando [create-event-source-mapping](#) agrega una asignación de orígenes de eventos con un destino de SQS en caso de error a MyFunction:

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/  
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```

El siguiente comando [UpdateEventSourceMapping](#) agrega un destino S3 en caso de error al origen de eventos asociado a la entrada uuid:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Para eliminar un destino, introduzca una cadena vacía como argumento del parámetro `destination-config`:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config ''
```

```
--destination-config '{"OnFailure": {"Destination": ""}}'
```

## Ejemplo de registro de invocación SNS y SQS

El siguiente ejemplo muestra lo que Lambda envía a un destino de tema de SNS o cola de SQS cuando se produce un error en la invocación de un origen de eventos de Kafka. Cada una de las claves en `recordsInfo` contiene el tema y la partición de Kafka, separados por un guion. Por ejemplo, para la clave `"Topic-0"`, `Topic` es el tema de Kafka, y `0` es la partición. Para cada tema y partición, puede usar los desplazamientos y los datos de las marcas de tiempo para buscar los registros de invocación originales.

```
{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      },
      "Topic-1": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
```

```

        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
    }
}
}
}

```

### Ejemplo de registro de invocación de un destino S3

Para los destinos de S3, Lambda envía todo el registro de invocación junto con los metadatos al destino. El siguiente ejemplo muestra lo que Lambda envía a un bucket de S3 de destino cuando se produce un error en la invocación de un origen de evento de Kafka. Además de todos los campos del ejemplo anterior para los destinos de SQS y SNS, el campo `payload` contiene el registro de invocación original en forma de cadena JSON de escape.

```

{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",

```

```
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
    },
    "Topic-1": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
    }
}
},
"payload": "<Whole Event>" // Only available in S3
}
```

### Tip

También se recomienda habilitar el control de versiones de S3 en el bucket de destino.

## Solución de errores de asignación de orígenes de eventos de Apache Kafka autoadministrado

En los siguientes temas se proporcionan consejos para solucionar errores y problemas que puedan surgir cuando utilice Apache Kafka autoadministrado con Lambda. Si se encuentra con un problema que no aparezca en esta lista, puede utilizar el botón Comentarios de esta página para notificarlo.

Para obtener más ayuda con la solución de problemas, visite el [Centro de conocimiento de AWS](#).

### Errores de autenticación y autorización

Si falta alguno de los permisos necesarios para consumir datos del clúster de Kafka, Lambda muestra uno de los siguientes mensajes de error en la asignación de orígenes de eventos en LastProcessingResult.

#### Mensajes de error

- [Error de autorización del clúster a Lambda](#)
- [Error de autenticación de SASL](#)

- [El servidor no pudo autenticar Lambda](#)
- [Lambda no ha podido autenticar el servidor](#)
- [El certificado o la clave privada proporcionados no son válidos](#)

### Error de autorización del clúster a Lambda

Para SASL/SCRAM o mTLS, este error indica que el usuario proporcionado no tiene todos los permisos de lista de control de acceso (ACL) de Kafka necesarios que se indican a continuación:

- Clúster DescribeConfigs
- Descripción del grupo
- Grupo de lectura
- Descripción del tema
- Tema de lectura

Cuando crea las ACL de Kafka con los permisos de `kafka-cluster` necesarios, debe especificar el tema y el grupo como recursos. El nombre del tema debe coincidir con el tema de la asignación de origen de eventos. El nombre del grupo debe coincidir con el UUID de la asignación de origen de eventos.

Después de agregar los permisos necesarios al rol de ejecución, pueden pasar varios minutos hasta que los cambios surtan efecto.

### Error de autenticación de SASL

Para SASL/SCRAM o SASL/PLAIN, este error indica que las credenciales de inicio de sesión proporcionadas no son válidas.

### El servidor no pudo autenticar Lambda

Este error indica que el agente de Kafka no ha podido autenticar Lambda. Este error puede producirse por cualquiera de las razones siguientes:

- No proporcionó ningún certificado de cliente para la autenticación de mTLS.
- Proporcionó un certificado de cliente, pero los agentes de Kafka no están configurados para utilizar la autenticación de mTLS.
- Los agentes de Kafka no confían en el certificado de cliente.

## Lambda no ha podido autenticar el servidor

Este error indica que Lambda no ha podido autenticar el agente de Kafka. Este error puede producirse por cualquiera de las razones siguientes:

- Los agentes de Kafka utilizan certificados autofirmados o una entidad de certificación privada, pero no proporcionaron el certificado de entidad de certificación raíz del servidor.
- El certificado de entidad de certificación raíz del servidor no coincide con la entidad de certificación raíz que firmó el certificado del agente.
- No se pudo validar el nombre de host porque el certificado del agente no contiene el nombre DNS ni la dirección IP del agente como nombre alternativo de sujeto.

El certificado o la clave privada proporcionados no son válidos

Este error indica que el consumidor de Kafka no ha podido utilizar la clave privada ni el certificado proporcionados. Asegúrese de que el formato del certificado y la clave sea PEM, y de que el cifrado de clave privada utilice un algoritmo PBES1.

## Errores de asignación de orígenes de eventos

Cuando agrega el clúster de Apache Kafka como un [origen de eventos](#) para su función de Lambda, si su función encuentra un error, su consumidor de Kafka deja de procesar registros. Los consumidores de una partición de tema son aquellos que se suscriben, leen y procesan sus registros. Sus otros consumidores de Kafka pueden continuar procesando registros, siempre que no encuentren el mismo error.

Para determinar la causa de un consumidor detenido, compruebe el `StateTransitionReason` campo en la respuesta de `EventSourceMapping`. En la siguiente lista se describen los errores de origen de eventos que puede recibir:

### **ESM\_CONFIG\_NOT\_VALID**

La configuración de asignación de orígenes de eventos no es válida.

### **EVENT\_SOURCE\_AUTHN\_ERROR**

Lambda no pudo autenticar el origen de eventos.

### **EVENT\_SOURCE\_AUTHZ\_ERROR**

Lambda no tiene los permisos necesarios para acceder al origen de eventos.

## FUNCTION\_CONFIG\_NOT\_VALID

La configuración de la función no es válida.

### Note

Si los registros de eventos de Lambda superan el límite de tamaño permitido de 6 MB, pueden no procesarse.



# Invocación de una función de Lambda mediante un punto de conexión de Amazon API Gateway

Puede crear una API web con un punto de enlace HTTP para la función Lambda utilizando Amazon API Gateway. API Gateway dispone de herramientas para crear y documentar API web que enrutan solicitudes HTTP a funciones de Lambda. Puede proteger el acceso a la API con controles de autenticación y autorización. Las API pueden atender el tráfico a través de Internet o estar accesibles exclusivamente en la VPC.

Los recursos de la API definen uno o varios métodos, como GET o POST. Los métodos tienen una integración que enruta las solicitudes a una función Lambda o a otro tipo de integración. Puede definir cada recurso y cada método de manera individual, o utilizar tipos de recursos y métodos especiales para que coincidan con todas las solicitudes que se ajustan a un patrón. Un [recurso proxy](#) captura todas las rutas que hay debajo de un recurso. El método ANY captura todos los métodos HTTP.

## Secciones

- [Elegir un tipo de API](#)
- [Adición de un punto de conexión a la función de Lambda](#)
- [Integración de proxy](#)
- [Formato de eventos](#)
- [Formato de respuesta](#)
- [Permisos](#)
- [Aplicación de muestra](#)
- [Tutorial: Uso de Lambda con API Gateway](#)
- [Manejo de los errores en Lambda con una API de API Gateway](#)

## Elegir un tipo de API

API Gateway admite tres tipos de API que invocan funciones de Lambda:

- [API HTTP](#): una API de RESTful ligera de baja latencia.
- [API de REST](#): una API de RESTful personalizable con gran variedad de características.
- [API de WebSocket](#): una API web que mantiene conexiones persistentes con clientes para una comunicación dúplex completa.

Las API HTTP y REST son API RESTful que procesan solicitudes HTTP y devuelven respuestas. Las API HTTP son más recientes y se construyen con la API de la versión 2 de API Gateway. Las siguientes características son nuevas en las API HTTP:

### Características de las API HTTP

- Implementaciones automáticas: cuando se modifican rutas o integraciones, los cambios se implementan automáticamente en etapas que tienen habilitada la implementación automática.
- Etapa predeterminada: puede crear una etapa predeterminada (`$default`) para atender las solicitudes en la ruta raíz de la URL de la API. En las etapas con nombre asignado, debe incluir el nombre de la etapa al principio de la ruta.
- Configuración CORS: puede configurar la API para agregar encabezados CORS a las respuestas salientes, en lugar de agregarlos en forma manual en el código de función.

Las API REST son las API RESTful clásicas que API Gateway admite desde su lanzamiento. En la actualidad, las API REST tienen más características de personalización, integración y administración.

### Características de las API REST

- Tipos de integración: las API REST son compatibles con las integraciones de Lambda personalizadas. Con una integración personalizada, puede enviar solo el cuerpo de la solicitud a la función o aplicar una plantilla de transformación al cuerpo de la solicitud antes de enviarla a la función.
- Control de acceso: las API REST admiten más opciones de autenticación y autorización.
- Monitoreo y seguimiento: la API REST admiten el seguimiento de AWS X-Ray y otras opciones de registro.

Para ver una comparativa detallada, consulte [Elección entre las API HTTP y las API REST](#) en la Guía para desarrolladores de API Gateway.

Las API de WebSocket también usan la API de la versión 2 de API Gateway y admiten un conjunto de características similares. Utilice una API de WebSocket para las aplicaciones que se benefician de una conexión persistente entre el cliente y la API. Las API de WebSocket proporcionan comunicación dúplex completa, lo que significa que tanto el cliente como la API pueden enviar mensajes continuamente sin esperar una respuesta.

Las API HTTP admiten un formato de evento simplificado (versión 2.0). En el ejemplo siguiente, se muestra el evento de una API HTTP.

Example [event-v2.json](#): evento de proxy API Gateway (API HTTP).

```
{
  "version": "2.0",
  "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
  "rawPath": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
  "rawQueryString": "",
  "cookies": [
    "s_fid=7AABXMPL1AFD9BBF-0643XMPL09956DE2",
    "regStatus=pre-register"
  ],
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "accept-encoding": "gzip, deflate, br",
    ...
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "r3pmxmplak",
    "domainName": "r3pmxmplak.execute-api.us-east-2.amazonaws.com",
    "domainPrefix": "r3pmxmplak",
    "http": {
      "method": "GET",
      "path": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
      "protocol": "HTTP/1.1",
      "sourceIp": "205.255.255.176",
      "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
    },
    "requestId": "JKJaXmPLvHcESHA=",
    "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
    "stage": "default",
    "time": "10/Mar/2020:05:16:23 +0000",
    "timeEpoch": 1583817383220
  },
  "isBase64Encoded": true
}
```

Para obtener más información, consulte este artículo sobre las [integraciones de AWS Lambda](#) en la Guía para desarrolladores de API Gateway.

## Adición de un punto de conexión a la función de Lambda

Para agregar un punto de enlace público a la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En Descripción general de la función, elija Agregar desencadenador.
4. Seleccione API Gateway.
5. Seleccione Create an API (Crear una API) o Use an existing API (Usar una API existente).
  - a. Nueva API: Para API type (Tipo de API), elija HTTP API (API HTTP). Para obtener más información, consulte [API types](#) (Tipos de API).
  - b. API existente: Selecciona la API en el menú desplegable o introduce el ID de API (p. ej., r3pmxmplak).
6. En Security (Seguridad), elija Open (Abrir).
7. Elija Añadir.

## Integración de proxy

Las API de API Gateway se componen de etapas, recursos, métodos e integraciones. La etapa y el recurso determinan la ruta del punto de enlace:

Formato de la ruta de las API

- /prod/: etapa prod y recurso raíz.
- /prod/user: etapa prod y recurso user.
- /dev/{proxy+}: cualquier ruta de la etapa dev.
- /: (API HTTP) etapa predeterminada y recurso raíz.

La integración de Lambda asigna una combinación de ruta y método HTTP a una función de Lambda. Puede configurar API Gateway para pasar el cuerpo de la solicitud HTTP tal cual (integración personalizada) o para encapsular el cuerpo de la solicitud en un documento que incluya toda la información de la solicitud, incluidos los encabezados, los recursos, la ruta y el método.

Para obtener más información, consulte [Configuración de las integraciones del proxy de Lambda en API Gateway](#).

## Formato de eventos

Amazon API Gateway invoca la función [sincrónicamente](#) con un evento que contiene una representación JSON de la solicitud HTTP. En las integraciones personalizadas, el evento es el cuerpo de la solicitud. En las integraciones de proxy, el evento tiene una estructura definida. En el siguiente ejemplo, se muestra un evento proxy procedente de una API REST de API Gateway.

Example [event.json](#) evento de proxy API Gateway (REST API)

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "GET",
  "requestContext": {
    "resourcePath": "/",
    "httpMethod": "GET",
    "path": "/Prod/",
    ...
  },
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "accept-encoding": "gzip, deflate, br",
    "Host": "70ixmpl4f1.execute-api.us-east-2.amazonaws.com",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-5e66d96f-7491f09xmpl179d18acf3d050",
    ...
  },
  "multiValueHeaders": {
    "accept": [
      "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
    ],
    "accept-encoding": [
      "gzip, deflate, br"
    ],
    ...
  },
  "queryStringParameters": null,
  "multiValueQueryStringParameters": null,
  "pathParameters": null,
  "stageVariables": null,
}
```

```
"body": null,  
"isBase64Encoded": false  
}
```

## Formato de respuesta

API Gateway espera una respuesta de la función y transmite el resultado a la persona que llama. En las integraciones personalizadas, debe definir una respuesta de integración y una respuesta de método para convertir la salida de la función en una respuesta HTTP. En las integraciones de proxy, la función debe responder con una representación de la respuesta en un formato específico.

En el ejemplo siguiente, se muestra un objeto response de una función Node.js. El objeto response representa una respuesta HTTP correcta que contiene un documento JSON.

Example [index.mjs](#): objeto de respuesta de integración de proxy (Node.js).

```
var response = {  
  "statusCode": 200,  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "isBase64Encoded": false,  
  "multiValueHeaders": {  
    "X-Custom-Header": ["My value", "My other value"],  
  },  
  "body": "{\n  \"TotalCodeSize\": 104330022,\n  \"FunctionCount\": 26\n}"  
}
```

El tiempo de ejecución de Lambda serializa el objeto response en un archivo JSON y lo envía a la API. La API analiza la respuesta y la utiliza para crear una respuesta HTTP, que luego envía al cliente que realizó la solicitud original.

## Example Respuesta HTTP

```
< HTTP/1.1 200 OK  
< Content-Type: application/json  
< Content-Length: 55  
< Connection: keep-alive  
< x-amzn-RequestId: 32998fea-xmpl-4268-8c72-16138d629356  
< X-Custom-Header: My value  
< X-Custom-Header: My other value
```

```
< X-Amzn-Trace-Id: Root=1-5e6aa925-ccecxmplbae116148e52f036
<
{
  "TotalCodeSize": 104330022,
  "FunctionCount": 26
}
```

## Permisos

Amazon API Gateway necesita permiso para invocar la función desde la [política basada en recursos](#) de la función. Puede conceder permiso de invocación a toda una API o conceder acceso limitado a una etapa, un recurso o un método.

Cuando agrega una API a la función utilizando la consola de Lambda, la consola de API Gateway o una plantilla de AWS SAM, la política basada en recursos de la función se actualiza automáticamente. A continuación se muestra una política de función de ejemplo.

### Example política de funciones

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLXE2F",
      "Effect": "Allow",
      "Principal": {
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:nodejs-apig-
function-1G3MXMPLXVXYI",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:execute-api:us-east-2:111122223333:ktyvxmls1/*/"
GET/"
        }
      }
    }
  ]
}
```

```
]
}
```

Puede administrar manualmente los permisos de las políticas de funciones con las siguientes operaciones de API:

- [AddPermission](#)
- [RemovePermission](#)
- [GetPolicy](#)

Para conceder permiso de invocación a una API existente, utilice el comando `add-permission`.

```
aws lambda add-permission --function-name my-function \
--statement-id apigateway-get --action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET/"
```

Debería ver los siguientes datos de salida:

```
{
  "Statement": [{"Sid": "apigateway-test-2", "Effect": "Allow", "Principal": {"Service": "apigateway.amazonaws.com"}, "Action": "lambda:InvokeFunction", "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function", "Condition": {"ArnLike": {"AWS:SourceArn": "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET"}}}]
}
```

#### Note

Si la función y la API se encuentran en Regiones de AWS diferentes, el identificador de región en el ARN del origen debe coincidir con la región de la función, no la región de la API. Cuando API Gateway invoca una función, utiliza el ARN de un recurso que se basa en el ARN de la API, pero que se ha modificado para que coincida con la región de la función.

El ARN de origen de este ejemplo concede permiso a una integración del método GET del recurso raíz en la etapa predeterminada de una API, con el ID `mnh1xmpli7`. Puede utilizar un asterisco en el ARN de origen para conceder permisos a varias etapas, métodos o recursos.



## Patrones de recursos

- `mnh1xmpli7/*/GET/*`: método GET de todos los recursos en todas las etapas.
- `mnh1xmpli7/prod/ANY/user`: método ANY del recurso `user` en la etapa `prod`.
- `mnh1xmpli7/**/*`: cualquier método de todos los recursos en todas las etapas.

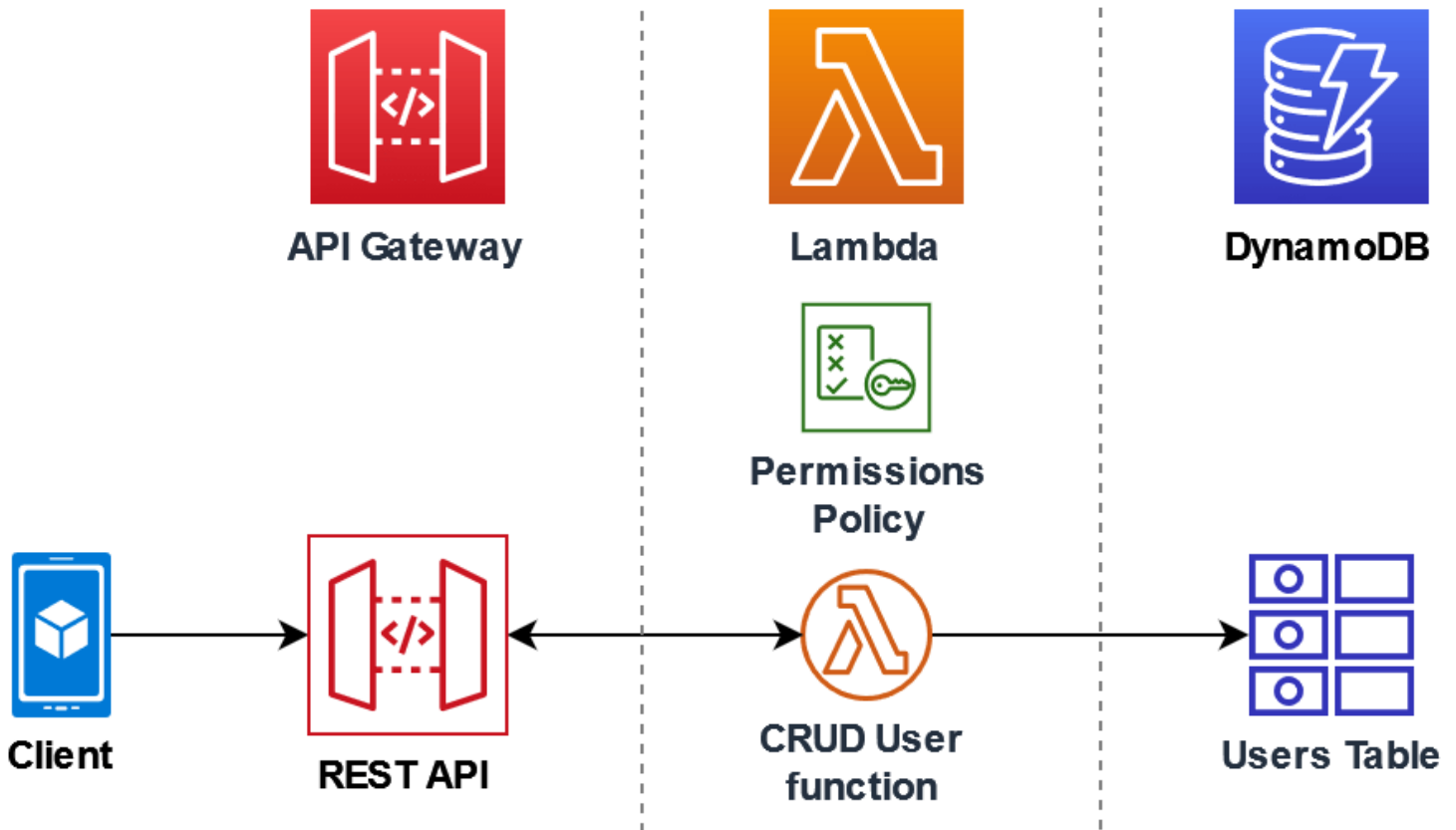
Para obtener información detallada acerca de cómo ver la política y quitar instrucciones, consulte [Trabajar con políticas de IAM basadas en recursos en Lambda](#).

## Aplicación de muestra

La aplicación de muestra [API Gateway con Node.js](#) incluye una función con una plantilla de SAM AWS SAM que crea una API de REST que tiene el seguimiento de AWS X-Ray habilitado. También incluye scripts para implementar, invocar la función, probar la API y limpiar.

## Tutorial: Uso de Lambda con API Gateway

En este tutorial, se crea una API de REST a través la cual invoca una función de Lambda mediante una solicitud HTTP. Su función de Lambda realizará operaciones de creación, lectura, actualización y eliminación (CRUD) en una tabla de DynamoDB. Se proporciona esta función a modo de demostración, pero aprenderá a configurar una API de REST de API Gateway que pueda invocar cualquier función de Lambda.



El uso de API Gateway proporciona a los usuarios un punto de conexión seguro en HTTP para invocar la función de Lambda y puede ayudar a gestionar grandes volúmenes de llamadas a su función al limitar el tráfico, validar y autorizar automáticamente las llamadas a la API. API Gateway también proporciona controles de seguridad flexibles mediante AWS Identity and Access Management (IAM) y Amazon Cognito. Esto resulta útil para los casos de uso en los que se requiere una autorización previa para las llamadas a la aplicación.

Para completar este tutorial, pasará por las siguientes etapas:

1. Cree y configure una función de Lambda en Python o Node.js para realizar operaciones en una tabla de DynamoDB.
2. Cree una API de REST en API Gateway para conectarse a la función de Lambda.
3. Cree una tabla de DynamoDB y pruébela con la función de Lambda en la consola.
4. Despliegue su API y pruebe la configuración completa con curl en una terminal.

Al completar estas etapas, aprenderá a utilizar API Gateway para crear un punto de conexión en HTTP que pueda invocar de forma segura una función de Lambda a cualquier escala. También

aprenderá a implementar su API, a probarla en la consola y a enviar una solicitud HTTP mediante una terminal.

## Secciones

- [Requisitos previos](#)
- [Crear una política de permisos](#)
- [Creación de un rol de ejecución](#)
- [Creación de la función](#)
- [Invoque la función de utilizando el comando de AWS CLI.](#)
- [Creación de una API REST mediante API Gateway](#)
- [Creación de un recurso en su API de REST.](#)
- [Creación de un método HTTP POST.](#)
- [Creación de una tabla de DynamoDB](#)
- [Pruebe la integración de API Gateway, Lambda y DynamoDB.](#)
- [Implementar la API](#)
- [Utilice curl para invocar tu función mediante solicitudes HTTP.](#)
- [Elimine sus recursos \(opcional\)](#)

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

### Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

### Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

## Instalar la AWS Command Line Interface

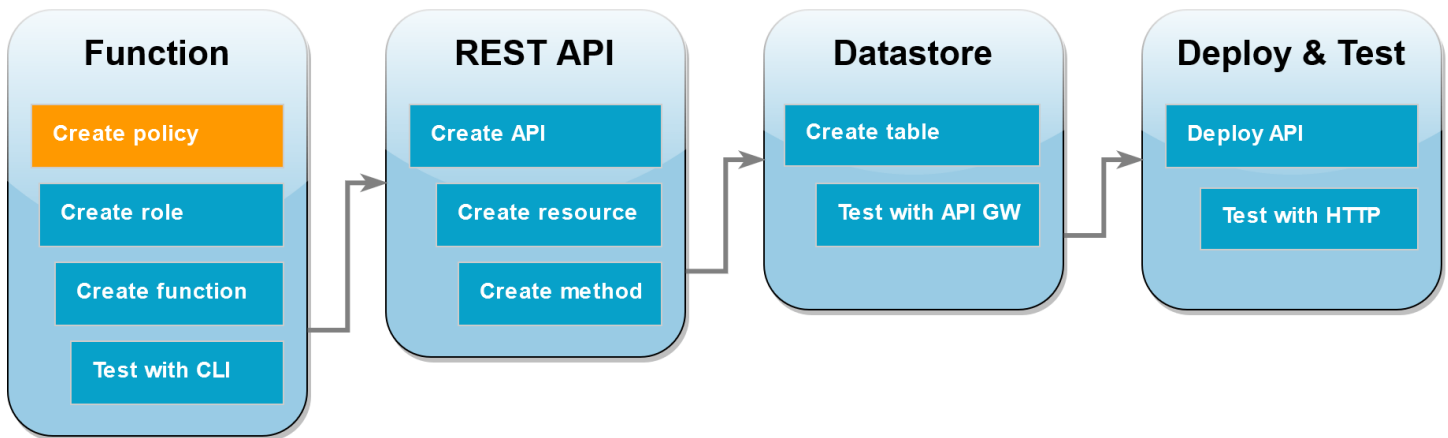
Si aún no ha instalado AWS Command Line Interface, siga los pasos que se indican en [Instalación o actualización de la versión más reciente de AWS CLI](#) para instalarlo.

El tutorial requiere un intérprete de comandos o un terminal de línea de comando para ejecutar los comandos. En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#).

## Crear una política de permisos



Para poder crear un [rol de ejecución](#) para la función de Lambda, primero debe crear una política de permisos que permita a la función acceder a los recursos de AWS necesarios. Para este tutorial, la política le permite a Lambda realizar operaciones de CRUD en una tabla de DynamoDB y escribir en Registros de Amazon CloudWatch.

Para crear la política de

1. Abra la página de [Políticas \(Políticas\)](#) de la consola de IAM.
2. Seleccione Crear política.
3. Elija la pestaña JSON y pegue la siguiente política personalizada en el editor JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1428341300017",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ],
}
```

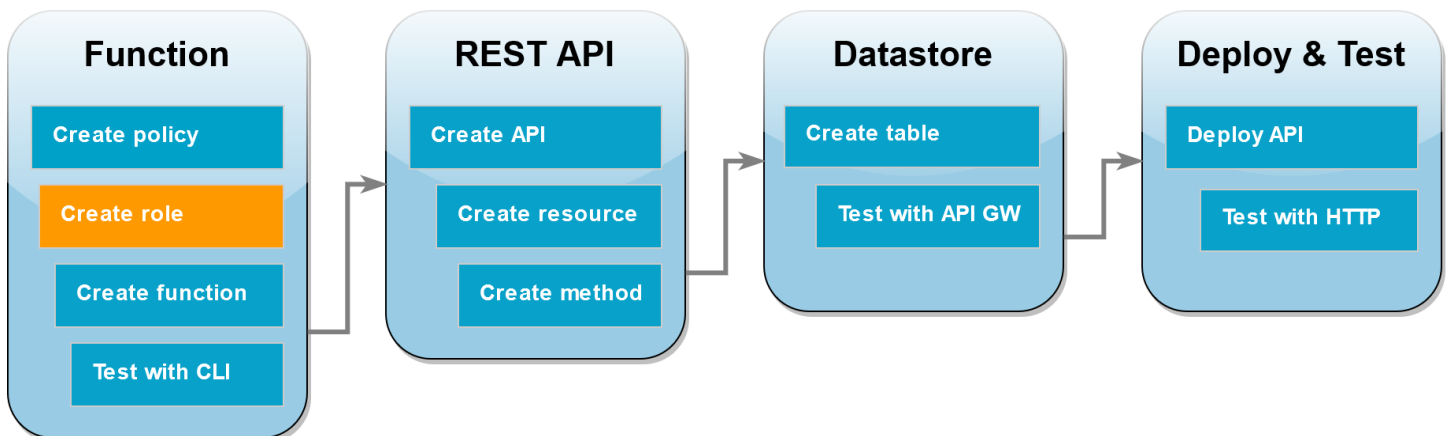
```

    "Sid": "",
    "Resource": "*",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Effect": "Allow"
  }
]
}

```

4. Elija Siguiente: Etiquetas.
5. Elija Siguiente: Revisar.
6. En Review policy (Revisar política), para el Name (Nombre) de la política, ingrese **lambda-apigateway-policy**.
7. Elija Crear política.

### Creación de un rol de ejecución



Un [rol de ejecución](#) es un rol de AWS Identity and Access Management (IAM) que concede a la función de Lambda permiso para acceder a servicios y recursos de AWS. Para permitir que la función realice operaciones en una tabla de DynamoDB, adjuntará la política de permisos que creó en el paso anterior.

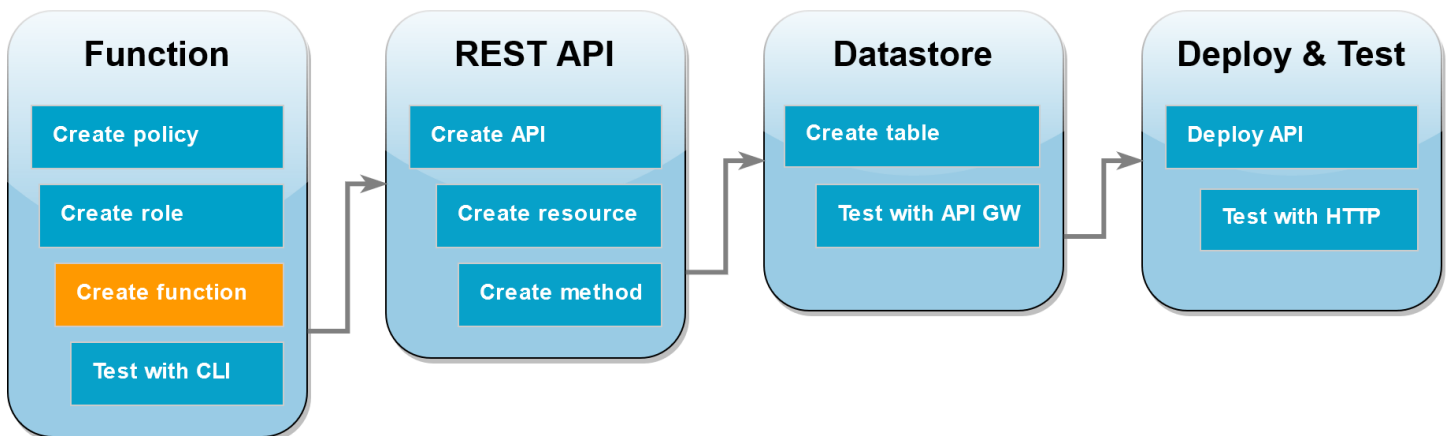
Para crear una función de ejecución y adjuntar su política de permisos personalizada

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.

3. Para el tipo de entidad de confianza, seleccione Servicio de AWS y, para el caso de uso, elija Lambda.
4. Elija Siguiente.
5. En el cuadro de búsqueda de políticas, escriba **lambda-apigateway-policy**.
6. En los resultados de búsqueda, seleccione la política que ha creado (**lambda-apigateway-policy**), y luego Next (Siguiente).
7. En Role details (Detalles del rol), introduzca **lambda-apigateway-role** en Role name (Nombre del rol) y, luego, elija Create role (Crear rol).

Más adelante en el tutorial, necesitará el nombre de recurso de Amazon (ARN) del rol que acaba de crear. En la página Roles (Roles) de la consola de IAM, seleccione el nombre de su rol (**lambda-apigateway-role**) y copie el Role ARN (ARN del rol) que aparece en la página Summary (Resumen).

## Creación de la función



El siguiente ejemplo de código recibe una entrada de evento de API Gateway en la que se especifica la operación que se debe realizar en la tabla de DynamoDB que creará y algunos datos de carga. Si los parámetros que recibe la función son válidos, realice la operación solicitada en la tabla.

## Node.js

### Example index.mjs

```
import { DynamoDBDocumentClient, PutCommand, GetCommand,
        UpdateCommand, DeleteCommand } from "@aws-sdk/lib-dynamodb";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```



```
const ddbClient = new DynamoDBClient({ region: "us-west-2" });
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);

// Define the name of the DDB table to perform the CRUD operations on
const tablename = "lambda-apigateway";

/**
 * Provide an event that contains the following keys:
 *
 * - operation: one of 'create,' 'read,' 'update,' 'delete,' or 'echo'
 * - payload: a JSON object containing the parameters for the table item
 *   to perform the operation on
 */
export const handler = async (event, context) => {

  const operation = event.operation;

  if (operation == 'echo'){
    return(event.payload);
  }

  else {
    event.payload.TableName = tablename;
    let response;

    switch (operation) {
      case 'create':
        response = await ddbDocClient.send(new PutCommand(event.payload));
        break;
      case 'read':
        response = await ddbDocClient.send(new GetCommand(event.payload));
        break;
      case 'update':
        response = ddbDocClient.send(new UpdateCommand(event.payload));
        break;
      case 'delete':
        response = ddbDocClient.send(new DeleteCommand(event.payload));
        break;
      default:
        response = 'Unknown operation: ${operation}';
    }
    console.log(response);
    return response;
  }
}
```

```
};
```

### Note

En este ejemplo, el nombre de la tabla de DynamoDB se define como una variable en el código de la función. En la aplicación real, la mejor práctica es pasar este parámetro como variable de entorno y evitar codificar el nombre de la tabla. Para obtener más información, consulte [Uso de variables de entorno de AWS Lambda](#).

## Cómo crear la función

1. Guarde el ejemplo de código como un archivo denominado `index.mjs` y, si es necesario, edite la región AWS especificada en el código. La región especificada en el código debe ser la misma que la región en la que se crea la tabla de DynamoDB más adelante en el tutorial.
2. Cree un paquete de despliegue utilizando el siguiente comando `zip`.

```
zip function.zip index.mjs
```

3. Cree una función de Lambda con el comando `create-function` de la AWS CLI. Para el parámetro `role`, introduzca el nombre de recurso de Amazon (ARN) del rol de ejecución que copió anteriormente.

```
aws lambda create-function \  
--function-name LambdaFunctionOverHttps \  
--zip-file fileb://function.zip \  
--handler index.handler \  
--runtime nodejs20.x \  
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

## Python 3

### Example LambdaFunctionOverHttps.py

```
import boto3  
  
# Define the DynamoDB table that Lambda will connect to  
table_name = "lambda-apigateway"
```

```
# Create the DynamoDB resource
dynamo = boto3.resource('dynamodb').Table(table_name)

# Define some functions to perform the CRUD operations
def create(payload):
    return dynamo.put_item(Item=payload['Item'])

def read(payload):
    return dynamo.get_item(Key=payload['Key'])

def update(payload):
    return dynamo.update_item(**{k: payload[k] for k in ['Key', 'UpdateExpression',
    'ExpressionAttributeNames', 'ExpressionAttributeValues'] if k in payload})

def delete(payload):
    return dynamo.delete_item(Key=payload['Key'])

def echo(payload):
    return payload

operations = {
    'create': create,
    'read': read,
    'update': update,
    'delete': delete,
    'echo': echo,
}

def lambda_handler(event, context):
    '''Provide an event that contains the following keys:
    - operation: one of the operations in the operations dict below
    - payload: a JSON object containing parameters to pass to the
    operation being performed
    ...'''

    operation = event['operation']
    payload = event['payload']

    if operation in operations:
        return operations[operation](payload)

    else:
        raise ValueError(f'Unrecognized operation "{operation}"')
```

**Note**

En este ejemplo, el nombre de la tabla de DynamoDB se define como una variable en el código de la función. En la aplicación real, la mejor práctica es pasar este parámetro como variable de entorno y evitar codificar el nombre de la tabla. Para obtener más información, consulte [Uso de variables de entorno de AWS Lambda](#).

## Cómo crear la función

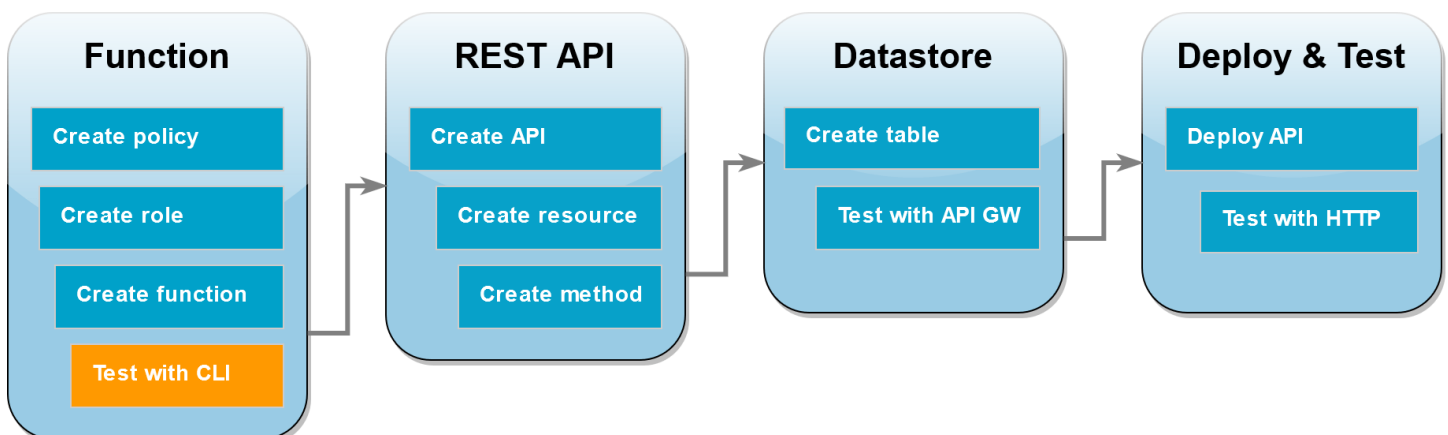
1. Guarde el código de ejemplo como un archivo denominado `LambdaFunctionOverHttps.py`.
2. Cree un paquete de despliegue utilizando el siguiente comando `zip`.

```
zip function.zip LambdaFunctionOverHttps.py
```

3. Cree una función de Lambda con el comando `create-function` de la AWS CLI. Para el parámetro `role`, introduzca el nombre de recurso de Amazon (ARN) del rol de ejecución que copió anteriormente.

```
aws lambda create-function \
--function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip \
--handler LambdaFunctionOverHttps.lambda_handler \
--runtime python3.12 \
--role arn:aws:iam::123456789012:role/service-role/Lambda-apigateway-role
```

Invoque la función de utilizando el comando de AWS CLI.



Antes de integrar la función con API Gateway, confirme que la ha implementado de forma correcta. Cree un evento de prueba que contenga los parámetros que la API de API Gateway enviará a Lambda y utilice el comando de la AWS CLI `invoke` para ejecutar la función.

Para invocar la función de Lambda con el comando AWS CLI.

1. Guarde el siguiente JSON como un archivo denominado `input.txt`.

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
}
```

2. Ejecute el siguiente comando AWS CLI de la `invoke`.

```
aws lambda invoke \
  --function-name LambdaFunctionOverHttps \
  --payload file://input.txt outputfile.txt \
  --cli-binary-format raw-in-base64-out
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

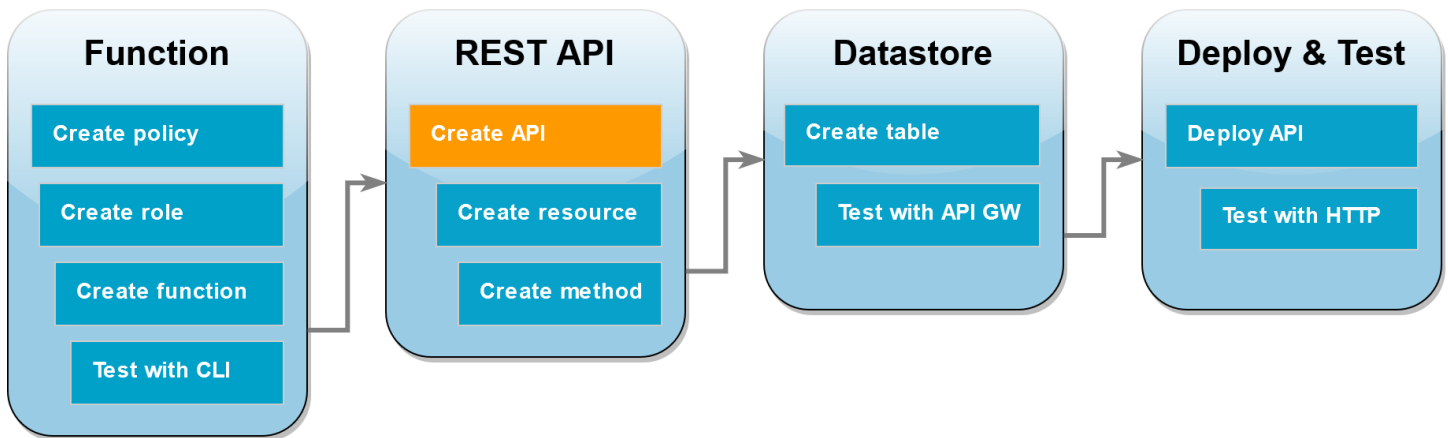
Debería ver la siguiente respuesta:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "LATEST"
}
```

3. Confirme que la función ha realizado la operación `echo` que especificó en el evento de prueba de JSON. Inspeccione el archivo `outputfile.txt` y compruebe que contiene lo siguiente:

```
{"somekey1": "somevalue1", "somekey2": "somevalue2"}
```

## Creación de una API REST mediante API Gateway

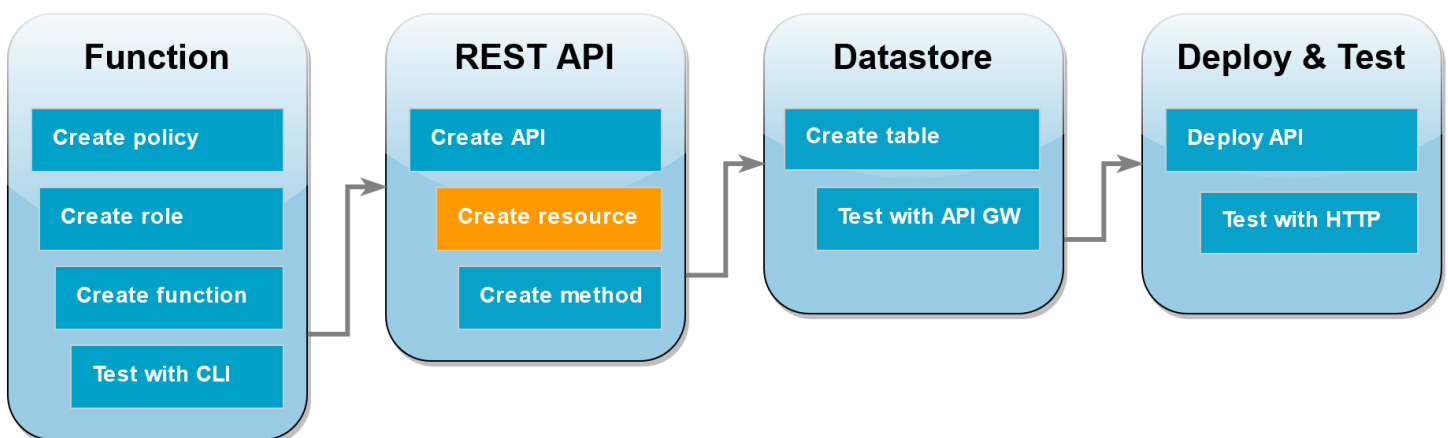


En este paso, crea la API de REST de API Gateway que utilizará para invocar la función de Lambda.

Para crear la API

1. Abra la [consola de API Gateway](#).
2. Seleccione Create API (Crear API).
3. En el cuadro REST API (API de REST), elija Build (Crear).
4. En Detalles de la API, deje seleccionada la opción Nueva API y, en Nombre de la API, ingrese **DynamoDBOperations**.
5. Seleccione Create API (Crear API).

Creación de un recurso en su API de REST.

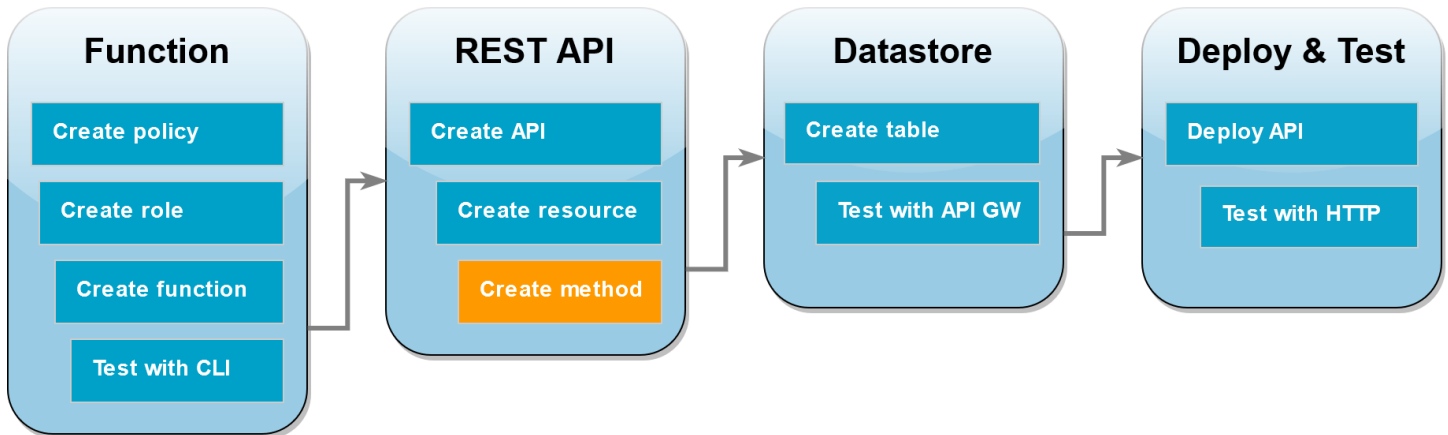


Para agregar un método HTTP a su API, primero debe crear un recurso para que funcione ese método. Aquí, se crea el recurso para administrar la tabla de DynamoDB.

## Para crear el recurso

1. En la [consola de puerta de enlace de API](#), en la página Recursos de su API, seleccione Crear recurso.
2. En Detalles del recurso, ingrese el nombre del recurso **DynamoDBManager**.
3. Elija Create Resource (Crear recurso).

## Creación de un método HTTP POST.



En este paso, creará un método (POST) para su recurso `DynamoDBManager`. Debe vincular este método POST a la función de Lambda para que cuando el método reciba una solicitud HTTP, API Gateway invoca la función de Lambda.

### **i** Note

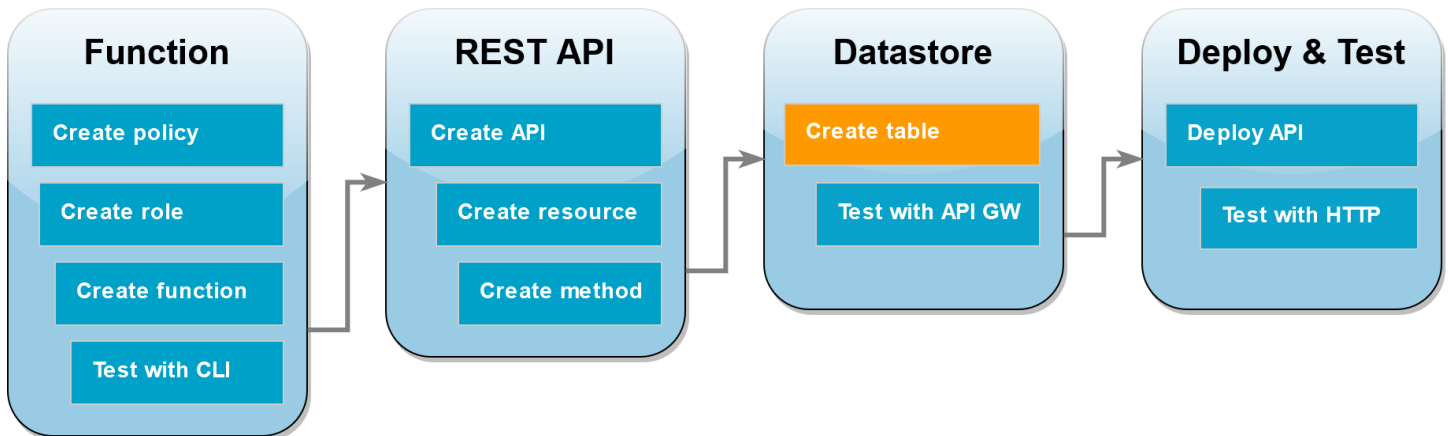
A los efectos de este tutorial, se utiliza un método HTTP (POST) para invocar una única función de Lambda que lleva a cabo todas las operaciones de la tabla de DynamoDB. En una aplicación real, la práctica recomendada es utilizar una función de Lambda y un método HTTP diferente para cada operación. Para obtener más información, consulte [The Lambda monolith](#) en Serverless Land.

## Para crear el método POST

1. En la página Recursos de su API, asegúrese de que el recurso `/DynamoDBManager` esté resaltado. A continuación, en el panel Métodos, seleccione Crear método.
2. En Tipo de método, elija POST.

3. En Tipo de integración, seleccione función de Lambda.
4. Para la función de Lambda, elija el nombre de recurso de Amazon (ARN) para la función (LambdaFunctionOverHttps).
5. Elija Crear método.

## Creación de una tabla de DynamoDB



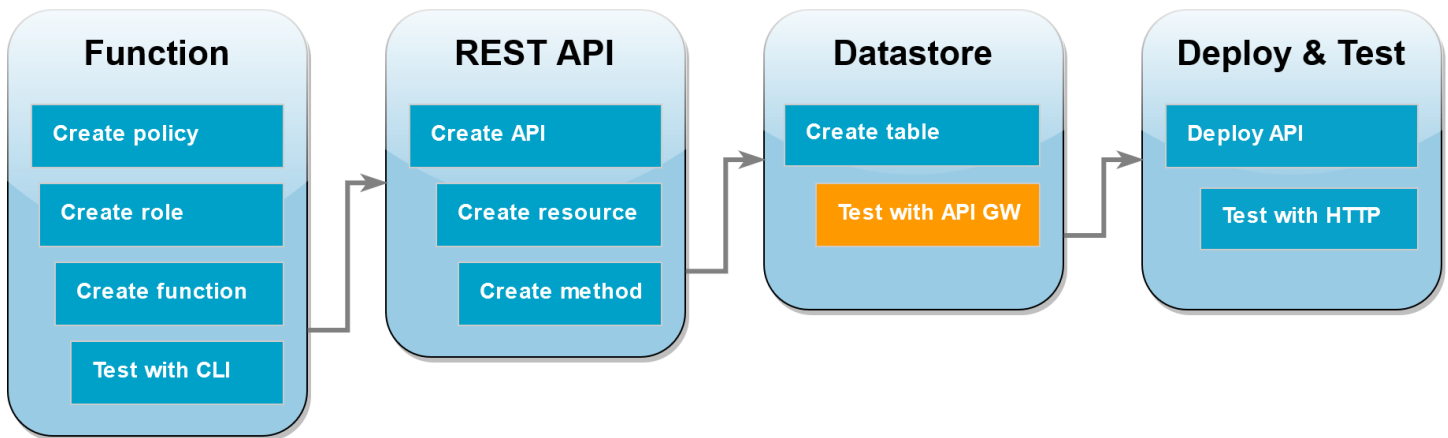
Cree una tabla de DynamoDB vacía en la que la función de Lambda realizará operaciones CRUD.

## Creación de la tabla de DynamoDB

1. Abra la [página Tables \(Tablas\) en la consola de DynamoDB](#).
2. Elija Crear tabla.
3. En Table details (Detalles de la tabla), haga lo siguiente:
  1. En Nombre de la tabla, introduzca **lambda-apigateway**.
  2. En Partition key (Clave de partición), ingrese **id** y mantenga el tipo de datos establecido como String (Cadena).
4. En Settings (Configuración), mantenga los valores predeterminados en Default settings (Configuración predeterminada).
5. Elija Crear tabla.



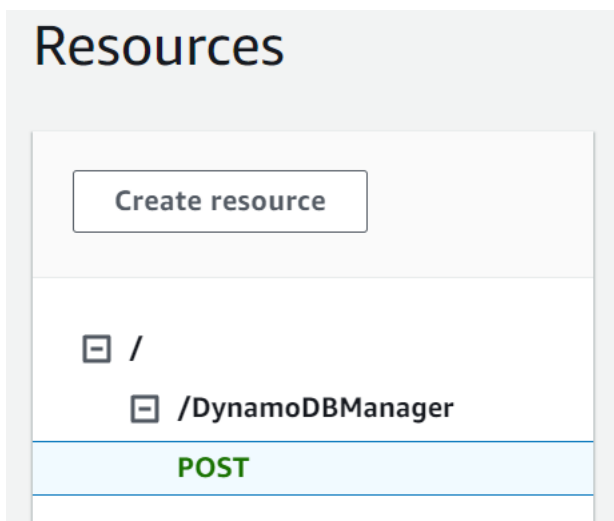
## Pruebe la integración de API Gateway, Lambda y DynamoDB.



Ya está listo para probar la integración del método de API de API Gateway con la función de Lambda y la tabla de DynamoDB. Con la consola de API Gateway, envíe las solicitudes directamente a su método POST mediante la función de prueba de la consola. En este paso, primero use una operación create para agregar un nuevo elemento a la tabla de DynamoDB y, a continuación, use una operación update para modificar el elemento.

### Prueba 1: Crear un nuevo elemento en la tabla de DynamoDB

1. En la [consola de API Gateway](#), elija su API (DynamoDBOperations).
2. Elija el método POST debajo del recurso DynamoDBManager.



3. Elija la pestaña Prueba. Puede que tenga que elegir el botón de flecha hacia la derecha para mostrar la pestaña.
4. En Método de prueba, deje vacías las cadenas de consulta y los encabezados. En Cuerpo de la solicitud, pegue el siguiente JSON:

```
{
  "operation": "create",
  "payload": {
    "Item": {
      "id": "1234ABCD",
      "number": 5
    }
  }
}
```

##### 5. Seleccione Probar.

Los resultados que se muestran al finalizar la prueba deben mostrar el estado 200. Este código de estado indica que la operación create se ha realizado correctamente.

Para confirmarlo, verifique que ahora su tabla de DynamoDB contenga un nuevo elemento.

6. Abra la página [Tables](#) (Tablas) en la consola de DynamoDB y elija la tabla `lambda-apigateway`.
7. Elija `Explore table items` (Explorar elementos de la tabla). En el panel `Items returned` (Devolución de elementos), debería ver un elemento con el `id1234ABCD` y el número de 5.

#### Prueba 2: Actualización del elemento de la tabla de DynamoDB

1. En la [consola de la puerta de enlace de API](#), vuelva a la pestaña Prueba del método POST.
2. En Método de prueba, deje vacías las cadenas de consulta y los encabezados. En Cuerpo de la solicitud, pegue el siguiente JSON:

```
{
  "operation": "update",
  "payload": {
    "Key": {
      "id": "1234ABCD"
    },
    "AttributeUpdates": {
      "number": {
        "Value": 10
      }
    }
  }
}
```

```
}

```

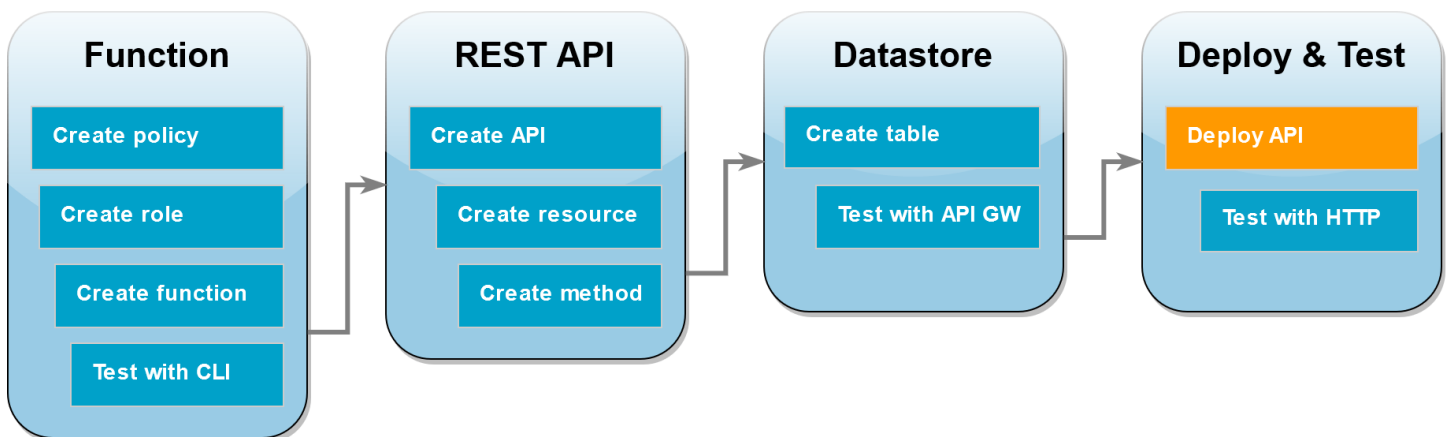
### 3. Seleccione Probar.

Los resultados que se muestran cuando finaliza la prueba deben mostrar el estado 200. Este código de estado indica que la operación de update se ha realizado correctamente.

Para confirmarlo, compruebe que el elemento de la tabla de DynamoDB se haya modificado.

- Abra la página [Tables](#) (Tablas) en la consola de DynamoDB y elija la tabla `lambda-apigateway`.
- Elija Explore table items (Explorar elementos de la tabla). En el panel Items returned (Devolución de elementos), debería ver un elemento con el `id1234ABCD` y el número de 10.

## Implementar la API



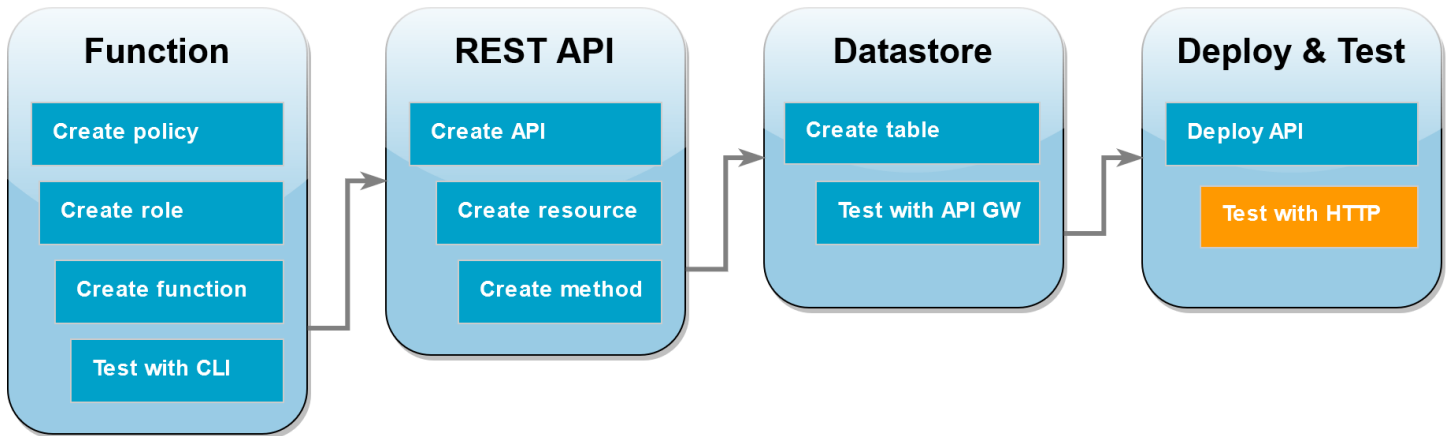
Para que un cliente llame a la API, debe crear una implementación y una etapa asociada. La etapa representa una imagen instantánea de su API, incluidos sus métodos e integraciones.

### Para implementar la API

- Abra la página API de la [consola de API Gateway](#) y seleccione la API de `DynamoDBOperations`.
- En la página Recursos de su API, seleccione Implementar la API.
- Para la Etapa de implementación, seleccione `*Nueva etapa*` y, en Nombre de etapa, introduzca **test**.
- Elija Implementar.

- En el panel Editor de etapas de prueba, copie la URL de invocación. Lo utilizará en el siguiente paso para invocar su función mediante una solicitud HTTP.

Utilice curl para invocar tu función mediante solicitudes HTTP.



Ahora puede invocar la función de Lambda al emitir una solicitud HTTP a la API. En este paso, creará un elemento nuevo en la tabla de DynamoDB y, a continuación, lo someterá a operaciones de lectura, actualización y eliminación.

Para crear un elemento en la tabla de DynamoDB con curl

- Ejecute el siguiente comando `curl` por medio de la URL de invocación que copió en el paso anterior. Cuando use el curl con la opción `-d` (data), utilice automáticamente el método HTTP POST.

```
curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number": 15}}}'
```

Si la operación se realizó correctamente, debería ver una respuesta con el código de estado HTTP 200.

- También puede utilizar la consola de DynamoDB para comprobar que el nuevo elemento esté en la tabla con las siguientes acciones:
  - Abra la página [Tables](#) (Tablas) en la consola de DynamoDB y elija la tabla `lambda-apigateway`.
  - Elija `Explore table items` (Explorar elementos de la tabla). En el panel `Items returned` (Devolución de elementos), debería ver un elemento con el `id5678EFGH` y el `número15`.

## Para leer el elemento en la tabla de DynamoDB con curl

- Ejecute el siguiente comando `curl` para leer el valor del elemento que acaba de crear. Utilice su propia URL de invocación.

```
curl https://avos4dr2rk.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager -d \
'{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

Debería ver uno de los siguientes resultados dependiendo de si eligió el código de función Node.js o Python:

### Node.js

```
{"$metadata":
{"statusCode":200,"requestId":"7BP3G5Q0C001E50FBQI9NS099JVV4KQNS05AEMVJF66Q9ASUAAJG",
"attempts":1,"totalRetryDelay":0},"Item":{"id":"5678EFGH","number":15}}
```

### Python

```
{"Item":{"id":"5678EFGH","number":15},"ResponseMetadata":
{"RequestId":"QNDJICE52E86B82VETR6RKBE5BVV4KQNS05AEMVJF66Q9ASUAAJG",
"HTTPStatusCode":200,"HTTPHeaders":{"server":"Server","date":"Wed, 31 Jul 2024
00:37:01 GMT","content-type":"application/x-amz-json-1.0",
"content-length":"52","connection":"keep-alive","x-amzn-
requestid":"QNDJICE52E86B82VETR6RKBE5BVV4KQNS05AEMVJF66Q9ASUAAJG","x-amz-
crc32":"2589610852"},
"RetryAttempts":0}}
```

## Para actualizar el elemento en la tabla de DynamoDB con curl

1. Ejecute el siguiente comando `curl` para actualizar el elemento que acaba de crear cambiando el valor `number`. Utilice su propia URL de invocación.

```
curl https://avos4dr2rk.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "update", "payload": {"Key": {"id": "5678EFGH"},
"UpdateExpression": "SET #num = :new_value", "ExpressionAttributeNames": {"#num":
"number"}, "ExpressionAttributeValues": {":new_value": 42}}}'
```

2. Para confirmar que se ha actualizado el valor de `number` correspondiente al elemento, ejecute otro comando de lectura:

```
curl https://avos4dr2rk.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

Para eliminar el elemento en la tabla de DynamoDB con `curl`

1. Si lo desea, ejecute el siguiente comando para eliminar el `curl` que ha creado. Utilice su propia URL de invocación.

```
curl https://l8togsqxd8.execute-api.us-west-2.amazonaws.com/test/DynamoDBManager \
-d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
```

2. Confirme que la operación de eliminación se ha realizado correctamente. En el panel Elementos devueltos de la página Explorar elementos de la consola de DynamoDB, verifique que el elemento con id 5678EFGH ya no se encuentre en la tabla.

## Elimine sus recursos (opcional)

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete(Eliminar).

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.

4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

Para eliminar la API

1. Abra la [página API](#) de la consola de API Gateway.
2. Seleccione la API que ha creado.
3. Elija Actions (Acciones), Delete (Eliminar).
4. Elija Eliminar.

Para eliminar una tabla de DynamoDB

1. Abra la página [Tables \(Tablas\)](#) en la consola de DynamoDB.
2. Seleccione la tabla que ha creado.
3. Elija Eliminar.
4. Escriba **delete** en el cuadro de texto.
5. Elija Delete table (Eliminar tabla).

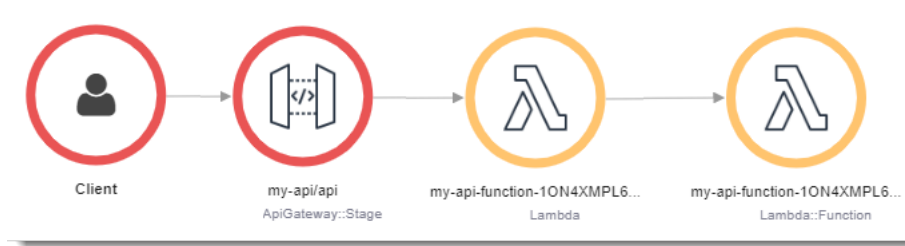
## Manejo de los errores en Lambda con una API de API Gateway

API Gateway trata todos los errores de las invocaciones y las funciones como errores internos. Si la API de Lambda rechaza la solicitud de invocación, API Gateway devuelve un código de error 500. Si la función se ejecuta pero devuelve un error o devuelve una respuesta con un formato incorrecto, API Gateway muestra un 502. En ambos casos, el cuerpo de la respuesta de API Gateway es `{"message": "Internal server error"}`.

### Note

API Gateway no vuelve a intentar ninguna invocación de Lambda. Si Lambda devuelve un error, API Gateway devuelve una respuesta de error al cliente.

En el ejemplo siguiente, se muestra un mapa de seguimiento de X-Ray de una solicitud que generó un error de función y un error 502 de API Gateway. El cliente recibirá el mensaje de error genérico.



Para personalizar la respuesta del error, debe detectar errores en el código y elaborar una respuesta en el formato que desee.

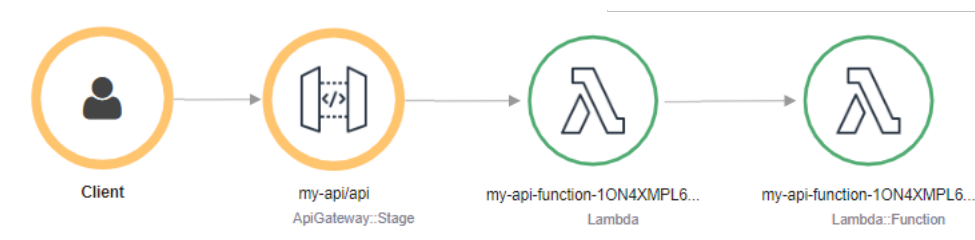
Example [index.mjs](#): error de formato.

```

var formatError = function(error){
  var response = {
    "statusCode": error.statusCode,
    "headers": {
      "Content-Type": "text/plain",
      "x-amzn-ErrorType": error.code
    },
    "isBase64Encoded": false,
    "body": error.code + ": " + error.message
  }
  return response
}

```

API Gateway convierte esta respuesta en un error HTTP con un código de estado y un cuerpo personalizados. En el mapa de seguimiento, el nodo de la función es verde porque el error se administró correctamente.



## Uso de AWS Lambda con AWS Application Composer

AWS Application Composer es un generador visual para diseñar aplicaciones modernas en AWS. Diseña la arquitectura de la aplicación arrastrando, agrupando y conectando Servicios de AWS en un



lienzo visual. Application Composer crea plantillas de infraestructura como código (IaC) a partir de su diseño que puede implementar mediante [AWS SAM](#) o [AWS CloudFormation](#).

## Exportación de la función de Lambda a Application Composer

Para empezar a utilizar Application Composer, cree un nuevo proyecto basado en la configuración de una función de Lambda existente mediante la consola de Lambda. Haga lo siguiente para exportar la configuración y el código de la función a Application Composer para crear un nuevo proyecto:

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Seleccione la función que desee utilizar como base para su proyecto de Application Composer.
3. En el panel Descripción general de la función, elija Exportar a Application Composer.

Para exportar la configuración y el código de la función a Application Composer, Lambda crea un bucket de Amazon S3 en su cuenta para almacenar estos datos temporalmente.

4. En el cuadro de diálogo, seleccione Confirmar y crear proyecto para aceptar el nombre predeterminado de este bucket y exportar la configuración y el código de la función a Application Composer.
5. (Opcional) Para elegir otro nombre para el bucket de Amazon S3 que Lambda crea, introduzca un nombre nuevo y elija Confirmar y crear proyecto. Los nombres de los buckets de Amazon S3 no pueden repetirse en ningún lado y deben seguir las [reglas de nomenclatura de buckets](#).
6. Para guardar los archivos de funciones y proyectos en Application Composer, active el [modo de sincronización local](#).

### Note

Si ya ha utilizado la característica Exportar a Application Composer y ha creado un bucket de Amazon S3 con el nombre predeterminado, Lambda puede volver a utilizar este bucket si aún existe. Acepte el nombre del bucket predeterminado en el cuadro de diálogo para volver a utilizar el bucket existente.

## Configuración del bucket de transferencia de Amazon S3

El bucket de Amazon S3 que Lambda crea para transferir la configuración de la función cifra automáticamente los objetos mediante el estándar de cifrado AES 256. Lambda también configura el

bucket para que utilice la [condición de propietario del bucket](#) para garantizar que solo su Cuenta de AWS pueda agregar objetos al bucket.

Lambda configura el bucket para eliminar automáticamente los objetos 10 días después de su carga. Sin embargo, Lambda no elimina automáticamente el bucket en sí. Para eliminar el bucket de su Cuenta de AWS, siga las instrucciones que se indican en [Eliminar un bucket](#). El nombre predeterminado del bucket usa el prefijo `lambdasam`, una cadena alfanumérica de 10 dígitos y el Región de AWS en la que creó la función:

```
lambdasam-06f22da95b-us-east-1
```

Para evitar que se le añadan cargos adicionales a su Cuenta de AWS, le recomendamos que elimine el bucket de Amazon S3 en cuanto termine de exportar su función a Application Composer.

Se aplican los [precios estándar de Amazon S3](#).

## Permisos necesarios

A fin de usar la característica de integración de Lambda con Application Composer, necesita ciertos permisos para descargar una plantilla AWS SAM y escribir la configuración de la función en Amazon S3.

Si quiere descargar una plantilla AWS SAM, debe tener permiso para utilizar las siguientes acciones de la API:

- [GetPolicy](#)
- [iam:GetPolicyVersion](#)
- [iam:GetRole](#)
- [iam:GetRolePolicy](#)
- [iam>ListAttachedRolePolicies](#)
- [iam>ListRolePolicies](#)
- [iam>ListRoles](#)

Puede otorgar permisos para utilizar todas estas acciones si agrega la política [AWSLambda\\_ReadOnlyAccess](#) administrada por AWS a su rol de usuario de IAM.

Para que Lambda escriba la configuración de su función en Amazon S3, debe tener permiso para usar las siguientes acciones de la API:

- [S3:PutObject](#)
- [S3:CreateBucket](#)
- [S3:PutBucketEncryption](#)
- [S3:PutBucketLifecycleConfiguration](#)

Si no puede exportar su configuración de la función a Application Composer, compruebe que su cuenta tiene los permisos necesarios para estas operaciones. Si tiene los permisos necesarios, pero aún no puede exportar la configuración de su función, compruebe si hay [Políticas basadas en recursos](#) que podrían limitar el acceso a Amazon S3.

## Otros recursos

Para obtener un tutorial más detallado sobre cómo diseñar una aplicación sin servidor en Application Composer basada en una función de Lambda existente, consulte [Infraestructura como código \(IaC\)](#).

Para utilizar Application Composer y AWS SAM para diseñar e implementar una aplicación sin servidor completa con Lambda, también puede seguir el [Tutorial de AWS Application Composer](#) en el [AWSTaller de patrones sin servidor](#).

## Uso de AWS Lambda con AWS CloudFormation

En una plantilla de AWS CloudFormation puede especificar una función de Lambda como destino de un recurso personalizado. Utilice los recursos personalizados para procesar parámetros, recuperar valores de configuración o llamar a otros servicios de AWS durante los eventos del ciclo de vida de la pila.

El ejemplo siguiente invoca una función definida en otra parte de la plantilla.

### Example - Definición de recurso personalizado

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

El token de servicio es el Nombre de recurso de Amazon (ARN) de la función que AWS CloudFormation invoca al crear, actualizar o eliminar la pila. También puede incluir propiedades adicionales como `FunctionName`, que AWS CloudFormation pasa a su función tal cual.

AWS CloudFormation invoca la función de Lambda de forma [asíncrona](#) con un evento que incluye una URL de devolución de llamada.

### Example - Evento de mensaje de AWS CloudFormation

```
{
  "RequestType": "Create",
  "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-processor-primer-14R0R2T3JKU66",
  "ResponseURL": "https://cloudformation-custom-resource-response-useast1.s3-us-east-1.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-1%3A123456789012%3Astack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456%7Cprimerinvoke%7C5d478078-13e9-baf0-464a-7ef285ecc786?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1555451971&Signature=28UijZePE5I4dvukKQqM%2F9Rf1o4%3D",
  "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",
  "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
```

```

    "LogicalResourceId": "primerinvoke",
    "ResourceType": "AWS::CloudFormation::CustomResource",
    "ResourceProperties": {
        "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-processor-primer-14R0R2T3JKU66",
        "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"
    }
}

```

La función es responsable de devolver una respuesta a la URL de devolución de llamada que indica el éxito o el error de la operación. Para ver la sintaxis de respuesta completa, consulte [Objetos de respuesta de recursos personalizados](#).

#### Example - Respuesta de recursos personalizados de AWS CloudFormation

```

{
    "Status": "SUCCESS",
    "PhysicalResourceId": "2019/04/18/[$LATEST]b3d1bfc65f19ec610654e4d9b9de47a0",
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
    "LogicalResourceId": "primerinvoke"
}

```

AWS CloudFormation proporciona una biblioteca llamada `cfn-response` que controla el envío de la respuesta. Si define su función dentro de una plantilla, puede solicitar la biblioteca por nombre. A continuación, AWS CloudFormation añade la biblioteca al paquete de implementación que crea para la función.

Si la función que utiliza un recurso personalizado tiene asociada una [interfaz de red elástica](#), agregue los siguientes recursos a la política de VPC, donde **region** es la región en la que se encuentra la función, sin guiones. Por ejemplo, `us-east-1` es `useast1`. Esto permitirá que el recurso personalizado responda a la URL de devolución de llamada que envía una señal de vuelta a la pila de AWS CloudFormation.

```

arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",

```

La función de ejemplo siguiente invoca una segunda función. Si la llamada se realiza sin error, la función envía una respuesta de operación correcta a AWS CloudFormation y la actualización de la

pila continúa. La plantilla utiliza el tipo de recurso [AWS::Serverless::Function](#) proporcionado por AWS Serverless Application Model.

### Example -Función de recursos personalizados

```
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs16.x
      InlineCode: |
        var aws = require('aws-sdk');
        var response = require('cfn-response');
        exports.handler = function(event, context) {
          // For Delete requests, immediately send a SUCCESS response.
          if (event.RequestType == "Delete") {
            response.send(event, context, "SUCCESS");
            return;
          }
          var responseStatus = "FAILED";
          var responseData = {};
          var functionName = event.ResourceProperties.FunctionName
          var lambda = new aws.Lambda();
          lambda.invoke({ FunctionName: functionName }, function(err, invokeResult) {
            if (err) {
              responseData = {Error: "Invoke call failed"};
              console.log(responseData.Error + ":\n", err);
            }
            else responseStatus = "SUCCESS";
            response.send(event, context, responseStatus, responseData);
          });
        };
      Description: Invoke a function to create a log stream.
      MemorySize: 128
      Timeout: 8
      Role: !GetAtt role.Arn
      Tracing: Active
```

Si la función que el recurso personalizado invoca no está definida en una plantilla, puede obtener el código fuente de `cfn-response` desde el [módulo `cfn-response`](#) en la Guía del usuario de AWS CloudFormation.

---

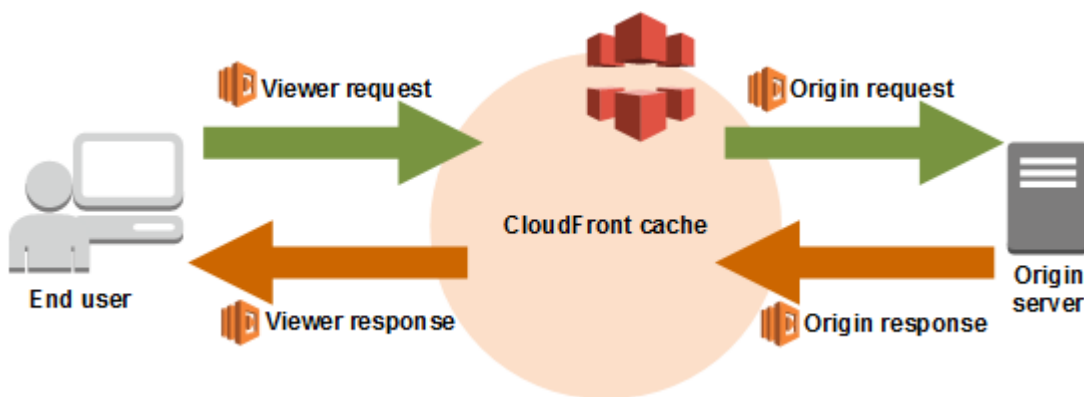
Para obtener más información sobre los recursos personalizados, consulte [Recursos personalizados](#) en la Guía del usuario de AWS CloudFormation.

## Uso AWS Lambda con CloudFront Lambda @Edge

[Lambda @Edge](#) es una extensión AWS Lambda que permite implementar funciones de Python y Node.js en las ubicaciones CloudFront perimetrales de Amazon. Un caso de uso común de Lambda @Edge consiste en utilizar funciones para personalizar el contenido que la CloudFront distribución entrega a los usuarios finales. Invocar estas funciones más cercanas al espectador en lugar de en servidores de origen reduce de forma significativa la latencia y mejora la experiencia del usuario.

Al asociar una CloudFront distribución a una función Lambda @Edge, CloudFront intercepta las solicitudes y respuestas en CloudFront las ubicaciones de borde. CloudFront a continuación, invoca la función Lambda mediante el envío de un evento. Puede hacer que CloudFront invoque la función Lambda cuando se produzcan los siguientes eventos:

- Cuando CloudFront recibe una solicitud de un espectador (solicitud del espectador)
- Antes CloudFront de reenviar una solicitud al origen (solicitud de origen)
- Cuando CloudFront recibe una respuesta del origen (respuesta de origen)
- Antes CloudFront devuelve la respuesta al espectador (respuesta del espectador)



### Note

Lambda@Edge admite un conjunto limitado de tiempos de ejecución y características. Para obtener más información, consulte [Requisitos y restricciones de las funciones Lambda](#) en la guía para CloudFront desarrolladores de Amazon.

El siguiente es un ejemplo de un CloudFront evento.



## Example CloudFront evento de mensaje

```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionId": "EDFDVBD6EXAMPLE"
        },
        "request": {
          "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",
          "method": "GET",
          "uri": "/picture.jpg",
          "headers": [
            {
              "key": "Host",
              "value": "d111111abcdef8.cloudfront.net"
            }
          ],
          "user-agent": [
            {
              "key": "User-Agent",
              "value": "curl/7.51.0"
            }
          ]
        }
      }
    }
  ]
}
```

Para obtener más información sobre el uso de Lambda @Edge, consulte [Uso con CloudFront Lambda @Edge](#).

# Procese eventos de Amazon DocumentDB con Lambda

Puede utilizar una función de Lambda para procesar eventos en un [flujo de cambios de Amazon DocumentDB \(con compatibilidad con MongoDB\)](#) si configura un clúster de Amazon DocumentDB como origen de eventos. A continuación, puede automatizar las cargas de trabajo basadas en eventos al invocar la función de Lambda cada vez que los datos cambien en su clúster de Amazon DocumentDB.

## Note

Lambda solo es compatible con la versión 4.0 y 5.0 de Amazon DocumentDB. Lambda no es compatible con la versión 3.6.

Asimismo, en el caso de las asignaciones de orígenes de eventos, Lambda solo es compatible con clústeres basados en instancias y clústeres regionales. Lambda no admite [clústeres elásticos](#) ni [clústeres globales](#). Esta limitación no se aplica cuando se utiliza Lambda como cliente para conectarse a Amazon DocumentDB. Lambda puede conectarse a todos los tipos de clústeres para realizar operaciones CRUD.

Lambda procesa los eventos de los flujos de cambios de Amazon DocumentDB de forma secuencial en el orden en que llegan. Por este motivo, la función solo puede gestionar una invocación simultánea desde DocumentDB a la vez. Para monitorear su función, puede realizar un seguimiento de sus [métricas de simultaneidad](#).

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

## Temas

- [Ejemplo de evento de Amazon DocumentDB](#)
- [Requisitos previos y permisos](#)

- [Configuración de red](#)
- [Creación de una asignación de orígenes de eventos de Amazon DocumentDB \(consola\)](#)
- [Creación de una asignación de orígenes de eventos de Amazon DocumentDB \(SDK o CLI\)](#)
- [Posiciones iniciales de flujos y sondeo](#)
- [Monitoreo del origen de eventos de Amazon DocumentDB](#)
- [Tutorial: Uso de AWS Lambda con Amazon DocumentDB Streams](#)

## Ejemplo de evento de Amazon DocumentDB

```
{
  "eventSourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:canaryclusterb2a659a2-qo5tcmqkc103",
  "events": [
    {
      "event": {
        "_id": {
          "_data": "0163eeb6e7000000090100000009000041e1"
        },
        "clusterTime": {
          "$timestamp": {
            "t": 1676588775,
            "i": 9
          }
        },
        "documentKey": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          }
        },
        "fullDocument": {
          "_id": {
            "$oid": "63eeb6e7d418cd98afb1c1d7"
          },
          "anyField": "sampleValue"
        },
        "ns": {
          "db": "test_database",
          "coll": "test_collection"
        },
        "operationType": "insert"
      }
    }
  ]
}
```

```
    }  
  }  
],  
  "eventSource": "aws:docdb"  
}
```

Para obtener más información sobre los eventos de este ejemplo y sus formas, consulte [Eventos de cambio](#) en el sitio web de la documentación de MongoDB.

## Requisitos previos y permisos

Antes de utilizar Amazon DocumentDB como un origen de eventos para su función de Lambda, tenga en cuenta los siguientes requisitos previos. Debe hacer lo siguiente:

- Tener un clúster de Amazon DocumentDB existente en la misma Cuenta de AWS y Región de AWS que su función. Si no dispone de un clúster existente, puede crearlo mediante los pasos de [Introducción a Amazon DocumentDB](#) en la Guía para desarrolladores de Amazon DocumentDB. Como alternativa, el primer conjunto de pasos en [Tutorial: Uso de AWS Lambda con Amazon DocumentDB Streams](#) lo guiará para crear un clúster de DocumentDB con todos los requisitos previos necesarios.
- Permitir que Lambda tenga acceso a los recursos de Amazon Virtual Private Cloud (Amazon VPC) asociados a su clúster de Amazon DocumentDB. Para obtener más información, consulte [Configuración de red](#).
- Habilitar TLS en su clúster de Amazon DocumentDB. Este es el valor predeterminado. Si deshabilita TLS, Lambda no podrá comunicarse con el clúster.
- Activar los flujos de cambios en su clúster de Amazon DocumentDB. Para obtener más información, consulte [Uso de flujos de cambios de Amazon DocumentDB](#) en la Guía para desarrolladores de Amazon DocumentDB.
- Proporcionar a Lambda las credenciales para acceder a su clúster de Amazon DocumentDB. Al configurar el origen de eventos, proporcione la clave de [AWS Secrets Manager](#) que contiene los detalles de autenticación (nombre de usuario y contraseña) necesarios para acceder al clúster. Para proporcionar esta clave durante la configuración, puede realizar uno de los siguientes procedimientos:
  - Si está utilizando la consola de Lambda para la configuración, proporcione la clave en el campo Clave del administrador de secretos.
  - Si está utilizando la AWS Command Line Interface (AWS CLI), introduzca esta clave en la opción `source-access-configurations`. Puede incluir esta opción con los

comandos [create-event-source-mapping](#) o [update-event-source-mapping](#). Por ejemplo:

```
aws lambda create-event-source-mapping \  
  ...  
  --source-access-configurations  
  '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-  
west-2:123456789012:secret:DocDBSecret-AbC4E6"}]' \  
  ...
```

- Conceder permisos a Lambda para administrar los recursos relacionados con el flujo de Amazon DocumentDB. Agregue los siguientes permisos de forma manual al [rol de ejecución](#) de su función:
  - [rds:DescribeDBClusters](#)
  - [rds:DescribeDBClusterParameters](#)
  - [rds:DescribeDBSubnetGroups](#)
  - [ec2:CreateNetworkInterface](#)
  - [ec2:DescribeNetworkInterfaces](#)
  - [ec2:DescribeVpcs](#)
  - [ec2>DeleteNetworkInterface](#)
  - [ec2:DescribeSubnets](#)
  - [ec2:DescribeSecurityGroups](#)
  - [kms:Decrypt](#)
  - [secretsmanager:GetSecretValue](#)
- Mantener el tamaño de los eventos del flujo de cambios de Amazon DocumentDB que envíe a Lambda por debajo de 6 MB. Lambda admite tamaños de carga de hasta 6 MB. Si el flujo de cambios intenta enviar a Lambda un evento de más de 6 MB, Lambda elimina el mensaje y emite la métrica `OversizedRecordCount`. Lambda emite todas las métricas dentro de lo posible.

#### Note

Si bien las funciones de Lambda suelen tener un límite de tiempo de espera máximo de 15 minutos, las asignaciones de orígenes de eventos para Amazon MSK, Apache Kafka autoadministrado, Amazon DocumentDB y Amazon MQ para ActiveMQ y RabbitMQ solo admiten funciones con límites de tiempo de espera máximos de 14 minutos. Esta restricción

garantiza que la asignación de orígenes de eventos pueda gestionar correctamente los errores y reintentos de las funciones.

## Configuración de red

Para que Lambda utilice su clúster de Amazon DocumentDB como origen de eventos, necesita acceder a la Amazon VPC en la que reside su clúster. Le recomendamos que implemente [puntos de conexión de VPC](#) AWS PrivateLink para que Lambda acceda a su VPC. Implemente un punto de conexión de VPC para Lambda y, si el clúster utiliza autenticación, implemente también un punto de conexión de VPC para Secrets Manager.

De manera alternativa, asegúrese de que la VPC asociada a su clúster de Amazon DocumentDB incluya una puerta de enlace de NAT por subred pública. Para obtener más información, consulte [the section called “Acceso a Internet para funciones de VPC”](#).

Si usa puntos de enlace de VPC, también debe configurarlos para [habilitar los nombres DNS privados](#).

Al crear una asignación de orígenes de eventos para un clúster de Amazon DocumentDB, Lambda comprueba si las interfaces de red elásticas (ENI) ya están presentes en las subredes y los grupos de seguridad de la VPC del clúster. Si Lambda encuentra ENI existentes, intenta reutilizarlos. De lo contrario, Lambda crea nuevos ENI para conectarse al origen de eventos e invocar la función.

### Note

Las funciones de Lambda siempre se ejecutan dentro de VPC propiedad del servicio de Lambda. El servicio mantiene estas VPC de forma automática, y no son visibles para los clientes. También puede conectar su función a una Amazon VPC. En cualquier caso, la configuración de VPC de la función no afecta la asignación de orígenes de eventos. Solo la configuración de la VPC del origen de eventos determina cómo Lambda se conecta al origen de eventos.

## Reglas de los grupos de seguridad de VPC

Configure los grupos de seguridad de Amazon VPC que contienen su clúster con las siguientes reglas (como mínimo):

- Reglas de entrada: permiten todo el tráfico en el puerto del clúster de Amazon DocumentDB para los grupos de seguridad especificados para el origen de eventos. Amazon DocumentDB utiliza el puerto 27017 de forma predeterminada.
- Reglas de salida: permiten todo el tráfico en el puerto 443 para todos los destinos. Permita todo el tráfico en el puerto del clúster de Amazon DocumentDB. Amazon DocumentDB utiliza el puerto 27017 de forma predeterminada.
- Si utiliza puntos de conexión de VPC en lugar de una puerta de enlace NAT, los grupos de seguridad asociados a los puntos de conexión de VPC deben permitir todo el tráfico entrante en el puerto 443 desde los grupos de seguridad del origen de eventos.

## Uso de los puntos de enlace de la VPC

Cuando utiliza puntos de conexión de VPC, las llamadas a la API para invocar su función se enrutan a través de estos puntos de conexión mediante los ENI. La entidad principal del servicio de Lambda debe llamar a `lambda:InvokeFunction` para cualquier función que utilice esos ENI.

De forma predeterminada, los puntos de conexión de VPC tienen políticas de IAM abiertas. La práctica recomendada es restringir estas políticas para permitir que solo entidades principales realicen las acciones necesarias mediante ese punto de conexión. Para garantizar que la asignación de orígenes de eventos pueda invocar la función de Lambda, la política de puntos de conexión de VPC debe permitir que el principio del servicio de Lambda llame a `lambda:InvokeFunction`. Restringir las políticas de puntos de conexión de VPC para permitir únicamente las llamadas a la API que se originen en su organización impide que la asignación de orígenes de eventos funcione correctamente.

En el siguiente ejemplo de políticas de puntos de conexión de VPC, se muestra cómo conceder el acceso necesario a los puntos de conexión de Lambda.

Example Política del punto de conexión de VPC: punto de conexión de Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      }
    }
  ]
}
```

```

    ],
    },
    "Resource": "*"
  }
]
}

```

Si su clúster de Amazon DocumentDB utiliza la autenticación, también puede restringir la política de puntos de conexión de VPC para el punto de conexión de Secrets Manager. Para llamar a la API de Secrets Manager, Lambda usa su rol de función, no la entidad principal de servicio de Lambda. En el siguiente ejemplo, se muestra una política del punto de conexión de Secrets Manager.

Example Política de punto de conexión de VPC: punto de conexión de Secrets Manager

```

{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "customer_function_execution_role_arn"
        ]
      },
      "Resource": "customer_secret_arn"
    }
  ]
}

```

## Creación de una asignación de orígenes de eventos de Amazon DocumentDB (consola)

Para configurar una función de Lambda que lea desde un flujo de cambios de un clúster de Amazon DocumentDB, cree una [asignación de orígenes de eventos](#). En esta sección, se describe cómo hacer esto desde la consola. Para obtener las instrucciones de AWS SDK y la AWS CLI, consulte [the section called “Creación de una asignación de orígenes de eventos de Amazon DocumentDB \(SDK o CLI\)”](#).

Para crear una asignación de orígenes de eventos de Amazon DocumentDB (consola)

1. Abra la página de [Funciones](#) en la consola de Lambda.



2. Elija el nombre de una función.
3. En Descripción general de la función, elija Agregar desencadenador.
4. En Configuración del desencadenador, seleccione DocumentDB de la lista desplegable.
5. Configure las opciones requeridas y luego elija Agregar.

Lambda admite las siguientes opciones para los orígenes de eventos de Amazon DocumentDB:

- Clúster de DocumentDB: seleccione un clúster de Amazon DocumentDB.
- Activar el desencadenador: elija si desea activar el desencadenador de forma inmediata. Si marca esta casilla, la función comenzará a recibir tráfico de forma inmediata del flujo de cambios de Amazon DocumentDB que se especificó al crear la asignación de orígenes de eventos. Recomendamos desmarcar la casilla para crear la asignación de orígenes de eventos en un estado desactivado para realizar pruebas. Tras la creación, puede activar la asignación de orígenes de eventos en cualquier momento.
- Nombre de base de datos: introduzca el nombre de una base de datos dentro del clúster a consumir.
- (Opcional) Nombre de la colección: ingrese el nombre de la colección dentro de la base de datos que se va a consumir. Si no especifica una colección, Lambda escucha todos los eventos de cada colección en la base de datos.
- Tamaño del lote: establezca el número máximo de mensajes que se recuperarán en un solo lote, con un máximo de 10 000. El tamaño predeterminado del lote es de 100.
- Posición inicial: elija la posición en el flujo desde la que desea comenzar a leer los registros.
  - Más recientes: procesa solo los registros nuevos que se agreguen al flujo principal. La función comienza a procesar registros solo después de que Lambda termina de crear el origen de eventos. Esto significa que es posible que se eliminen algunos registros hasta que el origen de eventos se haya creado correctamente.
  - Horizonte de supresión: procesar todos los registros del flujo. Lambda utiliza la duración de retención de registros de su clúster para determinar desde dónde comenzar a leer los eventos. En concreto, Lambda comienza a leer desde `current_time - log_retention_duration`. Su flujo de cambios ya debe estar activo antes de esta marca de tiempo para que Lambda lea de forma correcta todos los eventos.
  - En marca temporal: procesar los registros comenzando a partir de un momento determinado. Su flujo de cambios ya debe estar activo antes de la marca de tiempo especificada para que Lambda lea de forma correcta todos los eventos.

- Autenticación: elija el método de autenticación para acceder a los agentes en su clúster.
  - BASIC\_AUTH: con la autenticación básica, debe proporcionar la clave de Secrets Manager que contiene las credenciales para acceder al clúster.
- Clave de Secrets Manager: elija la clave de Secrets Manager que contiene los detalles de autenticación (nombre de usuario y contraseña) necesarios para acceder al clúster de Amazon DocumentDB.
- (Opcional) Intervalo de lote: la cantidad de tiempo máxima para recopilar registros antes de invocar la función, arriba de 300.
- (Opcional) Configuración completa del documento: para las operaciones de actualización de los documentos, elija lo que desea enviar al flujo. El valor predeterminado es Default, lo que significa que, para cada evento de flujo de cambios, Amazon DocumentDB envía solo un delta que describe los cambios realizados. Para obtener más información sobre este campo, consulte [FullDocument](#) en la documentación de la API de Javadocs para MongoDB.
  - Predeterminado: Lambda envía solo un documento parcial que describe los cambios realizados.
  - UpdateLookup: Lambda envía un delta que describe los cambios, junto con una copia del documento completo.

## Creación de una asignación de orígenes de eventos de Amazon DocumentDB (SDK o CLI)

Para crear o administrar una asignación de orígenes de eventos de Amazon DocumentDB con [AWS SDK](#), puede utilizar las siguientes operaciones de la API:

- [CreateEventSourceMapping](#)
- [ListEventSourceMappings](#)
- [GetEventSourceMapping](#)
- [UpdateEventSourceMapping](#)
- [DeleteEventSourceMapping](#)

Para crear la asignación de orígenes de eventos con la AWS CLI, use el comando [create-event-source-mapping](#). En el siguiente ejemplo, se utiliza este comando para asignar una función denominada my-function a un flujo de cambios de Amazon DocumentDB. El origen del evento se especifica mediante un nombre de recurso de Amazon (ARN) con un tamaño de lote de 500, que comienza desde una marca temporal en tiempo Unix. El comando también especifica

la clave de Secrets Manager que Lambda utiliza para conectarse a Amazon DocumentDB. Además, incluye parámetros `document-db-event-source-config` que especifican la base de datos y la colección desde la que se va a leer.

```
aws lambda create-event-source-mapping --function-name my-function \  
  --event-source-arn arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-  
epzcyvu4pjoy \  
  --batch-size 500 \  
  --starting-position AT_TIMESTAMP \  
  --starting-position-timestamp 1541139109 \  
  --source-access-configurations \  
  '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-  
east-1:123456789012:secret:DocDBSecret-BAtjxi"}]' \  
  --document-db-event-source-config '{"DatabaseName":"test_database",  
"CollectionName": "test_collection"}' \  

```

Debería ver un resultado con un aspecto similar al siguiente:

```
{  
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",  
  "BatchSize": 500,  
  "DocumentDBEventSourceConfig": {  
    "CollectionName": "test_collection",  
    "DatabaseName": "test_database",  
    "FullDocument": "Default"  
  },  
  "MaximumBatchingWindowInSeconds": 0,  
  "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-  
epzcyvu4pjoy",  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
  "LastModified": 1541348195.412,  
  "LastProcessingResult": "No records processed",  
  "State": "Creating",  
  "StateTransitionReason": "User action"  
}
```

Tras la creación, puede utilizar el comando [update-event-source-mapping](#) para actualizar la configuración relacionada con el origen de eventos de Amazon DocumentDB. El siguiente comando actualiza el tamaño del lote a 1000 y el intervalo del lote a 10 segundos. Para este comando, necesita el UUID de la asignación de orígenes de eventos, que puede recuperar mediante el comando `list-event-source-mapping` o desde la consola de Lambda.

```
aws lambda update-event-source-mapping --function-name my-function \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--batch-size 1000 \  
--batch-window 10
```

Debería ver esta salida con un aspecto similar al siguiente:

```
{  
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",  
  "BatchSize": 500,  
  "DocumentDBEventSourceConfig": {  
    "CollectionName": "test_collection",  
    "DatabaseName": "test_database",  
    "FullDocument": "Default"  
  },  
  "MaximumBatchingWindowInSeconds": 0,  
  "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-  
epzcyvu4pjoy",  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
  "LastModified": 1541359182.919,  
  "LastProcessingResult": "OK",  
  "State": "Updating",  
  "StateTransitionReason": "User action"  
}
```

Lambda actualiza la configuración de forma asíncrona, por lo que es posible que no vea estos cambios en la salida hasta que se complete el proceso. Para ver la configuración actual de la asignación de orígenes de eventos, utilice el comando [get-event-source-mapping](#).

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Debería ver esta salida con un aspecto similar al siguiente:

```
{  
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",  
  "DocumentDBEventSourceConfig": {  
    "CollectionName": "test_collection",  
    "DatabaseName": "test_database",  
    "FullDocument": "Default"  
  },  
  "BatchSize": 1000,
```

```
"MaximumBatchingWindowInSeconds": 10,  
"EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-  
epzcyvu4pjoy",  
"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
"LastModified": 1541359182.919,  
"LastProcessingResult": "OK",  
"State": "Enabled",  
"StateTransitionReason": "User action"  
}
```

Para eliminar la asignación de orígenes de eventos de Amazon DocumentDB, utilice el comando [delete-event-source-mapping](#).

```
aws lambda delete-event-source-mapping \  
--uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

## Posiciones iniciales de flujos y sondeo

Tenga en cuenta que el sondeo de flujos durante la creación y las actualizaciones de la asignación de orígenes de eventos es, en última instancia, coherente.

- Durante la creación de la asignación de orígenes de eventos, es posible que se demore varios minutos en iniciar el sondeo de los eventos del flujo.
- Durante las actualizaciones de la asignación de orígenes de eventos, es posible que se demore varios minutos en detener y reiniciar el sondeo de los eventos del flujo.

Este comportamiento significa que, si especifica LATEST como posición inicial del flujo, la asignación de orígenes de eventos podría omitir eventos durante la creación o las actualizaciones. Para garantizar que no se pierda ningún evento, especifique la posición inicial del flujo como TRIM\_HORIZON o AT\_TIMESTAMP.

## Monitoreo del origen de eventos de Amazon DocumentDB

Para ayudarlo a monitorear el origen de eventos de Amazon DocumentDB, Lambda emite la métrica `IteratorAge` cuando su función termina de procesar un lote de registros. La antigüedad del iterador es la diferencia entre la marca de tiempo del evento más reciente y la marca de tiempo actual. En esencia, la métrica `IteratorAge` indica la antigüedad del último registro procesado del lote. Si la función está procesando nuevos eventos en este momento, puede utilizar la antigüedad del iterador para estimar la latencia entre cuando un registro se agrega y cuando la función lo procesa.

Una tendencia ascendente en `IteratorAge` puede indicar problemas con la función. Para obtener más información, consulte [Ver las métricas de funciones de Lambda](#).

Los flujos de cambios de Amazon DocumentDB no están optimizados para gestionar grandes intervalos de tiempo entre eventos. Si su origen de eventos de Amazon DocumentDB no recibe ningún evento durante un periodo prolongado, Lambda puede deshabilitar la asignación de orígenes de eventos. La duración de este periodo puede variar de unas semanas a unos meses, según el tamaño del clúster y otras cargas de trabajo.

Lambda admite cargas de hasta 6 MB. Sin embargo, los eventos del flujo de cambios de Amazon DocumentDB pueden tener un tamaño de hasta 16 MB. Si el flujo de cambios intenta enviar a Lambda un evento de flujo de cambios de más de 6 MB, Lambda elimina el mensaje y emite la métrica `OversizedRecordCount`. Lambda emite todas las métricas dentro de lo posible.

## Tutorial: Uso de AWS Lambda con Amazon DocumentDB Streams

En este tutorial, creará una función de Lambda básica que consuma eventos de un flujo de cambios de Amazon DocumentDB (con compatibilidad con MongoDB). Para completar este tutorial, pasará por las siguientes etapas:

- Configure su clúster de Amazon DocumentDB, conéctese a él y active los flujos de cambios en él.
- Cree la función de Lambda y configure el clúster de Amazon DocumentDB como origen de eventos para su función.
- Para probar la configuración integral, inserte elementos en la base de datos de Amazon DocumentDB.

### Temas

- [Requisitos previos](#)
- [Crear el entorno AWS Cloud9](#)
- [Crear el grupo de seguridad de EC2](#)
- [Crear el clúster de DocumentDB](#)
- [Crear un secreto en Secrets Manager](#)
- [Instale el shell de mongo](#)
- [Conectarse al clúster de DocumentDB](#)
- [Activar flujos de cambios](#)
- [Creación de puntos de conexión de VPC de interfaz](#)

- [Creación del rol de ejecución](#)
- [Creación de la función de Lambda](#)
- [Crear la asignación de orígenes de eventos de Lambda](#)
- [Probar la función: invocación manual](#)
- [Probar la función: insertar un registro](#)
- [Probar la función: actualizar un registro](#)
- [Probar la función: eliminar un registro](#)
- [Eliminación de sus recursos](#)

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

## Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

## Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

## Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.



Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

## Instalar la AWS Command Line Interface

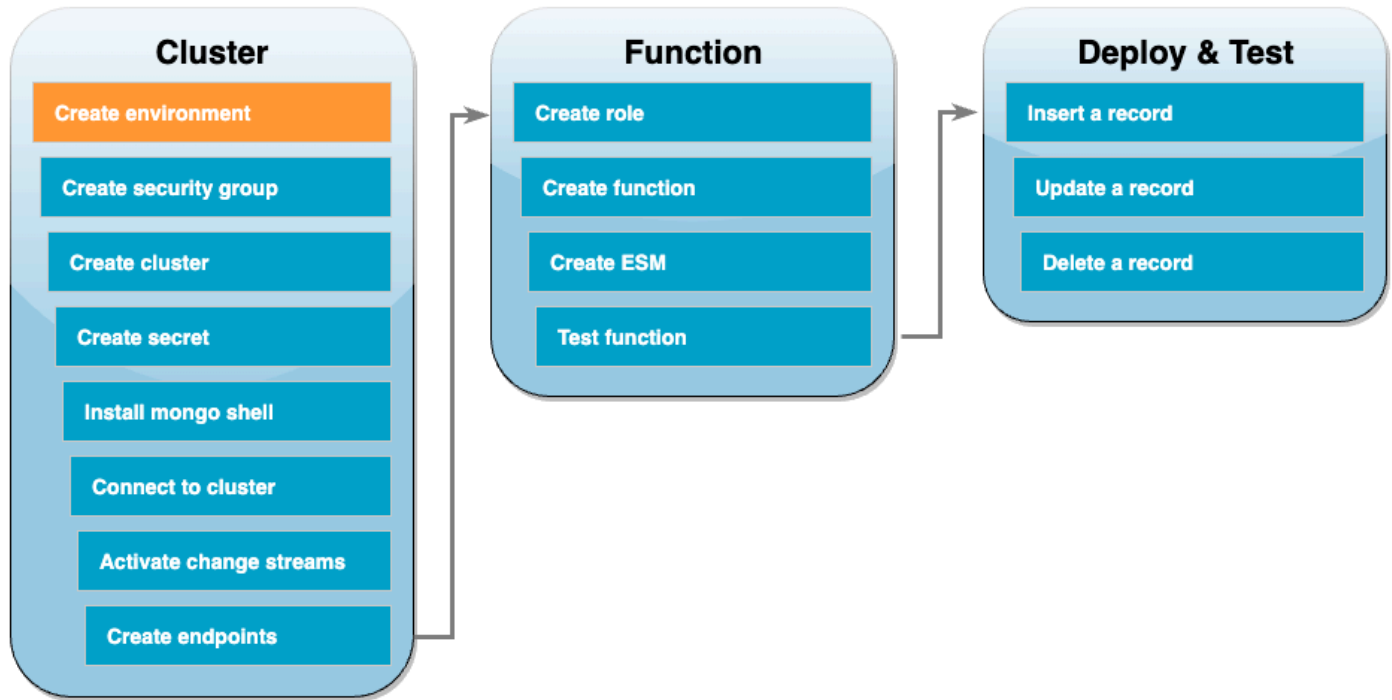
Si aún no ha instalado AWS Command Line Interface, siga los pasos que se indican en [Instalación o actualización de la versión más reciente de AWS CLI](#) para instalarlo.

El tutorial requiere un intérprete de comandos o un terminal de línea de comando para ejecutar los comandos. En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#).

## Crear el entorno AWS Cloud9



Antes de crear la función de Lambda, debe crear y configurar el clúster de Amazon DocumentDB. Los pasos para configurar el clúster de este tutorial se basan en el procedimiento de [Introducción a Amazon DocumentDB](#).

### **i** Note

Si ya tiene configurado un clúster de Amazon DocumentDB, asegúrese de activar los flujos de cambios y crear los puntos de conexión de VPC de interfaz necesarios. Luego puede pasar directamente a los pasos de creación de la función.

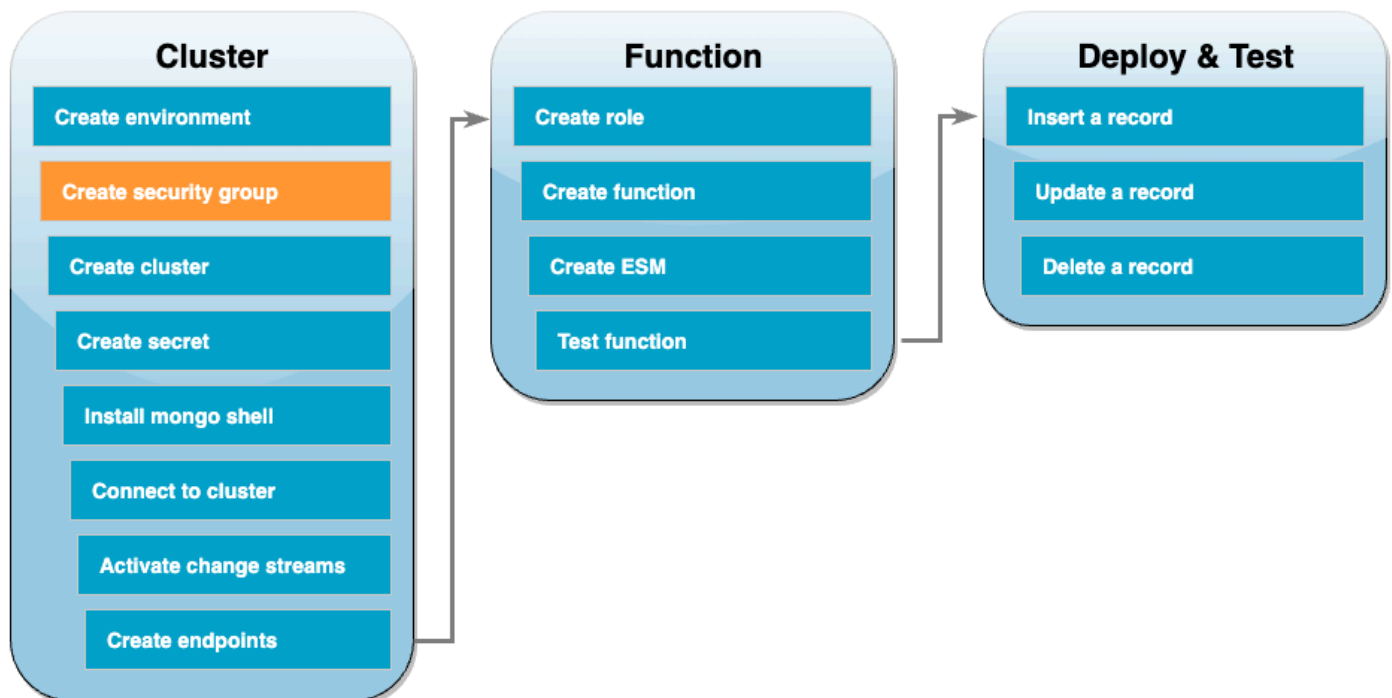
Primero, cree un entorno AWS Cloud9. Utilizará este entorno a lo largo de este tutorial para conectarse a su clúster de DocumentDB y realizar consultas en él.

Para crear un entorno de AWS Cloud9, realice los siguientes pasos:

1. Abra la [consola de Cloud9](#) y elija Crear entorno.
2. Cree un entorno con la siguiente configuración:
  - En Detalles:

- Nombre: DocumentDBCloud9Environment
  - Tipo de entorno: nueva instancia de EC2
  - En Nueva instancia de EC2:
    - Tipo de instancia: t2.micro (1 GiB de RAM + 1 vCPU)
    - Plataforma: Amazon Linux 2
    - Tiempo de espera: 30 minutos
  - En Configuración de red:
    - Conexión: AWS Systems Manager (SSM)
    - Expanda el menú desplegable Configuración de VPC.
    - Nube virtual privada (VPC) de Amazon: elija su [VPC predeterminada](#).
    - Subred: sin preferencia
  - Conserve todas las otras opciones de configuración predeterminadas.
3. Seleccione Crear. Aprovisionar el nuevo entorno de AWS Cloud9 puede tardar varios minutos.

## Crear el grupo de seguridad de EC2

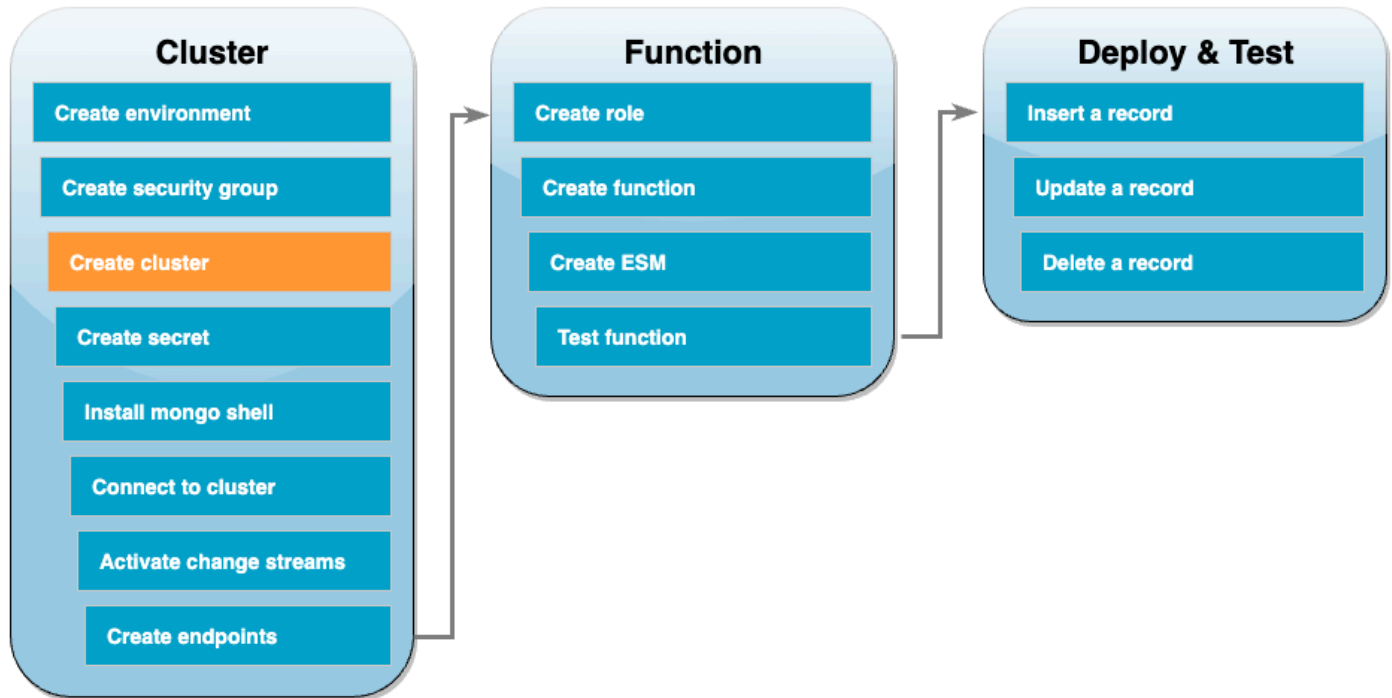


A continuación, cree un [grupo de seguridad de EC2](#) con reglas que permitan el tráfico entre el clúster de DocumentDB y el entorno de Cloud9.

## Para crear un grupo de seguridad de EC2

1. Abra la [consola de EC2](#). En Red y seguridad, elija Grupos de seguridad.
2. Elija Crear grupo de seguridad.
3. Cree un grupo de seguridad con la siguiente configuración:
  - En Detalles básicos:
    - Nombre del grupo de seguridad: DocDBTutorial
    - Descripción: grupo de seguridad para el tráfico entre Cloud9 y DocumentDB.
    - VPC: seleccione la [VPC predeterminada](#).
  - En Inbound rules (Reglas de entrada), elija Add rule (Agregar regla). Cree una regla con la siguiente configuración:
    - Tipo: TCP personalizado
    - Intervalo de puertos: 27017
    - Fuente: Custom (Personalizada)
    - En el cuadro de búsqueda situado junto a Origen, elija el grupo de seguridad para el entorno de AWS Cloud9 que creó en el paso anterior. Para ver una lista de los grupos de seguridad disponibles, ingrese c1oud9 en el cuadro de búsqueda. Elija el grupo de seguridad con el nombre aws-c1oud9-<environment\_name>.
  - Conserve todas las otras opciones de configuración predeterminadas.
4. Elija Crear grupo de seguridad.

## Crear el clúster de DocumentDB



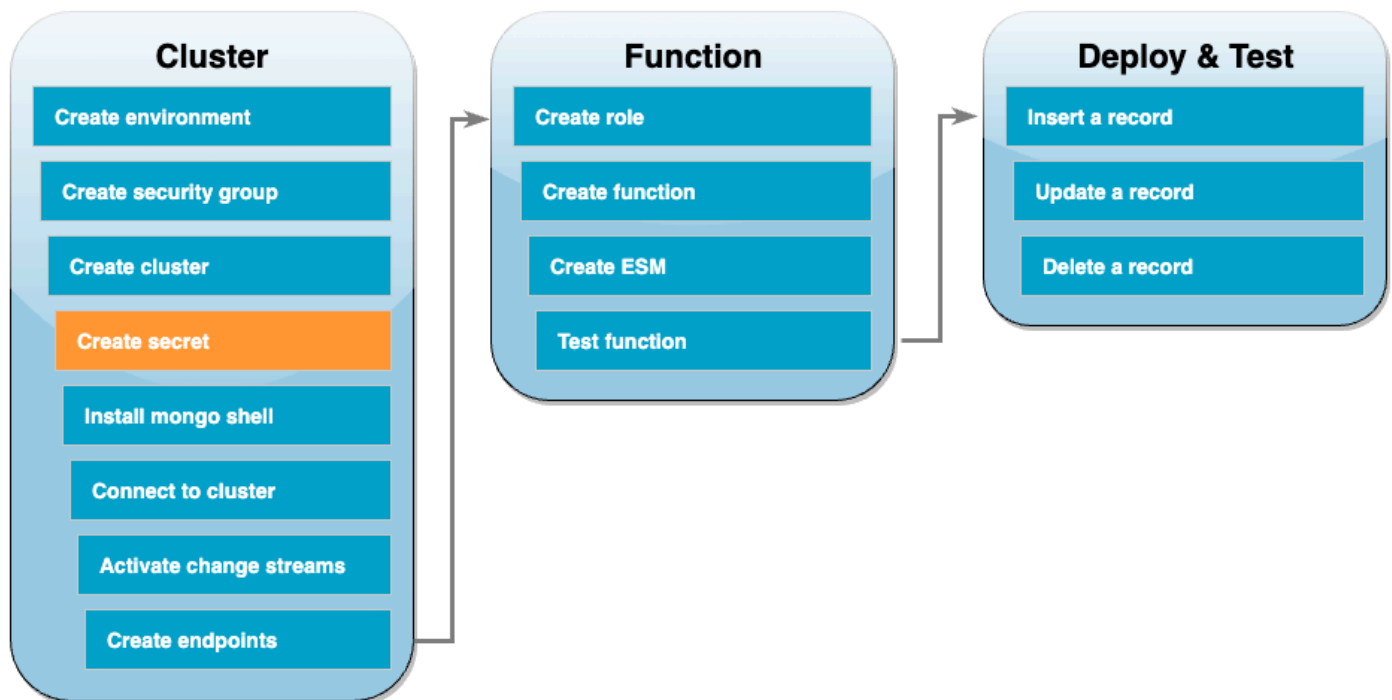
En este paso, creará un clúster de DocumentDB con el grupo de seguridad del paso anterior.

Para crear un clúster de DocumentDB

1. Abra la [consola de DocumentDB](#). En Clústeres, elija Crear.
2. Cree un clúster con la siguiente configuración:
  - En Tipo de clúster, elija Clúster basado en instancias.
  - En Configuración:
    - Versión del motor: 5.0.0
    - Clase de instancia: db.t3.medium (apta para prueba gratuita)
    - Número de instancias: 1.
  - En Autenticación:
    - Introduzca el Nombre de usuario y la Contraseña necesarios para conectarse al clúster (las mismas credenciales que utilizó para crear el secreto en el paso anterior). En Confirmar contraseña, confirme la contraseña.
  - Active Mostrar configuración avanzada.
  - En Configuración de red:

- Nube virtual privada (VPC): elija su [VPC predeterminada](#).
  - Grupo de subred: predeterminado
  - Grupos de seguridad de VPC: además de default (VPC), elija el grupo de seguridad DocDBTutorial (VPC) que creó en el paso anterior.
  - Conserve todas las otras opciones de configuración predeterminadas.
3. Elija Create cluster. Aprovisionar el clúster de DocumentDB puede tardar varios minutos.

## Crear un secreto en Secrets Manager



Para acceder al clúster de DocumentDB de forma manual, debe proporcionar las credenciales de nombre de usuario y contraseña. Para que Lambda acceda al clúster, debe proporcionar un secreto de Secrets Manager que contenga estas mismas credenciales de acceso al configurar la asignación de orígenes de eventos. En este paso, creará este secreto.

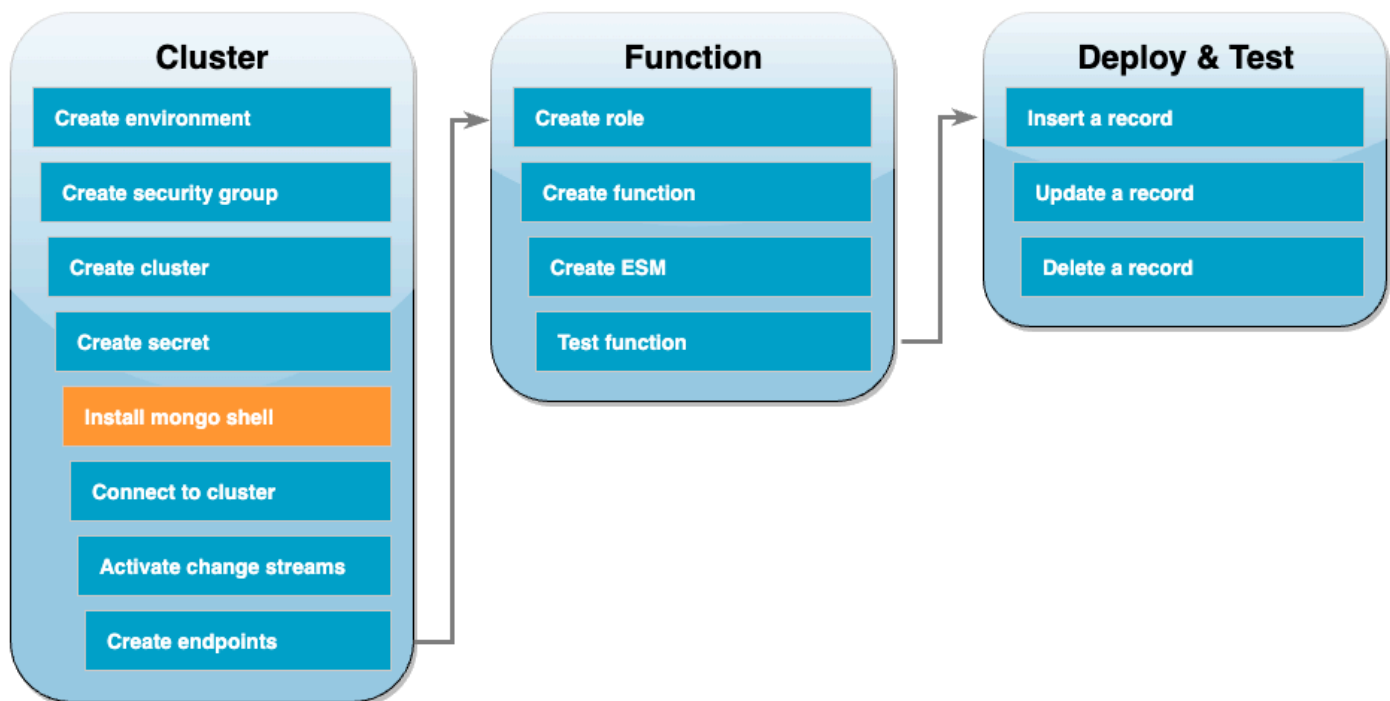
Para crear un secreto en Secrets Manager

1. Abra la consola de [Secrets Manager](#) y elija Almacenar un nuevo secreto.
2. En Elegir tipo de secreto, elija las siguientes opciones:
  - En Detalles básicos:

- Tipo de secreto: credenciales para la base de datos de Amazon DocumentDB
  - En Credenciales, ingrese el nombre de usuario y la contraseña que utilizará para acceder al clúster de DocumentDB.
  - Base de datos: elija su clúster de DocumentDB.
  - Elija Siguiente.
3. En Configurar secreto, elija las siguientes opciones:
    - Nombre del secreto: DocumentDBSecret
    - Elija Siguiente.
  4. Elija Siguiente.
  5. Elija Almacenar.
  6. Actualice la consola para comprobar que ha guardado correctamente el secreto DocumentDBSecret.

Anote el ARN secreto de su secreto. Lo necesitará en un paso posterior.

## Instale el shell de mongo



En este paso, instalará el intérprete de comandos de mongo en su entorno de Cloud9. El intérprete de comandos de mongo es una utilidad de línea de comandos que se utiliza para conectarse al clúster de DocumentDB y consultarlo.

Para instalar el intérprete de comandos de mongo en su entorno de Cloud9

1. Abra la [consola de Cloud9](#). Junto al entorno DocumentDBCloud9Environment que creó anteriormente, haga clic en el enlace Abrir en la columna IDE de Cloud9.
2. En la ventana de terminal, cree el archivo de repositorio de MongoDB con el siguiente comando:

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://\nrepo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86_64/\nngpgcheck=1 \nenabled=1\nngpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc" | sudo tee /etc/\nyum.repos.d/mongodb-org-5.0.repo
```

3. A continuación, instale el intérprete de comandos de mongo con el siguiente comando:

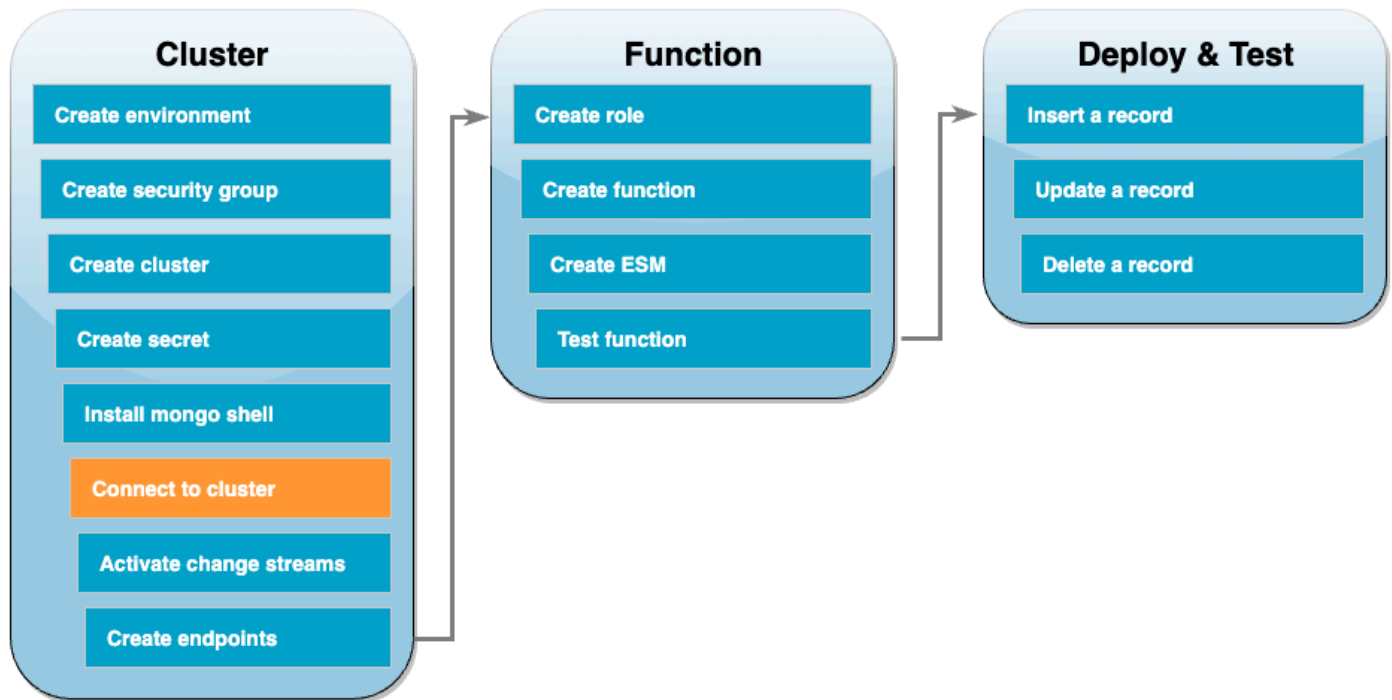
```
sudo yum install -y mongodb-org-shell
```

4. Para cifrar los datos en tránsito, descargue la [clave pública de Amazon DocumentDB](#). El siguiente comando descarga un archivo denominado `global-bundle.pem`:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```



## Conectarse al clúster de DocumentDB



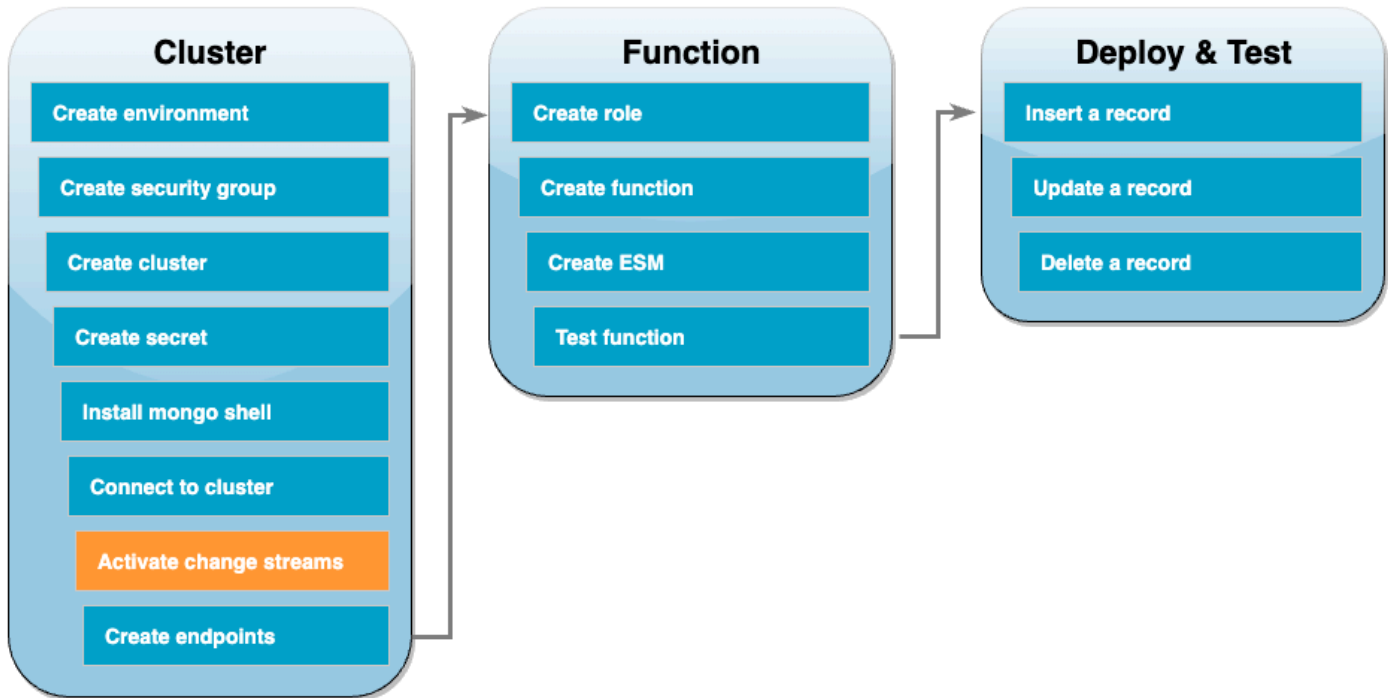
Ya está listo para conectarse a su clúster de DocumentDB mediante el intérprete de comandos de mongo.

Para conectarse al clúster de DocumentDB

1. Abra la [consola de DocumentDB](#). En Clústeres, elija su clúster al seleccionar su identificador de clúster.
2. En la pestaña Conectividad y seguridad, en Conectar a este clúster con el shell mongo, elija Copiar.
3. En su entorno de Cloud9, pegue este comando en el terminal. Sustituya `<insertYourPassword>` por la contraseña correcta.

Tras introducir este comando, si en el símbolo del sistema aparece `rs0:PRIMARY>`, está conectado a su clúster de Amazon DocumentDB.

## Activar flujos de cambios



En este tutorial, realizará un seguimiento de los cambios en la colección `products` de la base de datos `docdbdemo` en el clúster de DocumentDB. Para ello, active los [flujos de cambios](#). Primero, cree la base de datos `docdbdemo` e inserte un registro para probarla.

Para crear una nueva base de datos dentro del clúster

1. En su entorno de Cloud9, asegúrese de seguir [conectado al clúster de DocumentDB](#).
2. En la ventana de terminal, use el siguiente comando para crear una nueva base de datos llamada `docdbdemo`:

```
use docdbdemo
```

3. Luego use el siguiente comando para insertar un registro en `docdbdemo`:

```
db.products.insert({"hello":"world"})
```

Debería ver un resultado con un aspecto similar al siguiente:

```
WriteResult({ "nInserted" : 1 })
```

#### 4. Utilice el siguiente comando para enumerar todas las bases de datos:

```
show dbs
```

Asegúrese de que el resultado contenga la base de datos docdbdemo:

```
docdbdemo 0.000GB
```

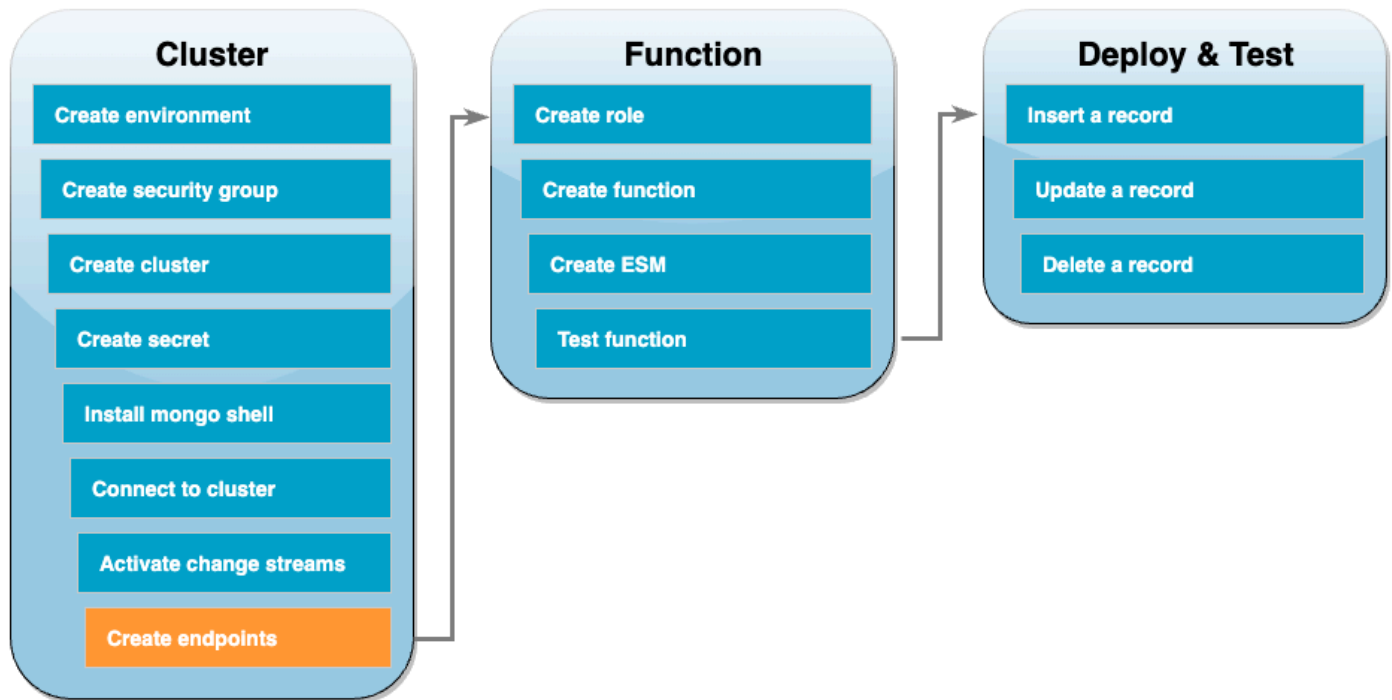
A continuación, active los flujos de cambios en la colección `products` de la base de datos `docdbdemo` con el siguiente comando:

```
db.adminCommand({modifyChangeStreams: 1,  
  database: "docdbdemo",  
  collection: "products",  
  enable: true});
```

Debería ver un resultado con un aspecto similar al siguiente:

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

## Creación de puntos de conexión de VPC de interfaz



A continuación, cree [puntos de conexión de VPC de interfaz](#) para garantizar que Lambda y Secrets Manager (que se utilizarán más adelante para almacenar nuestras credenciales de acceso al clúster) puedan conectarse a su VPC predeterminada.

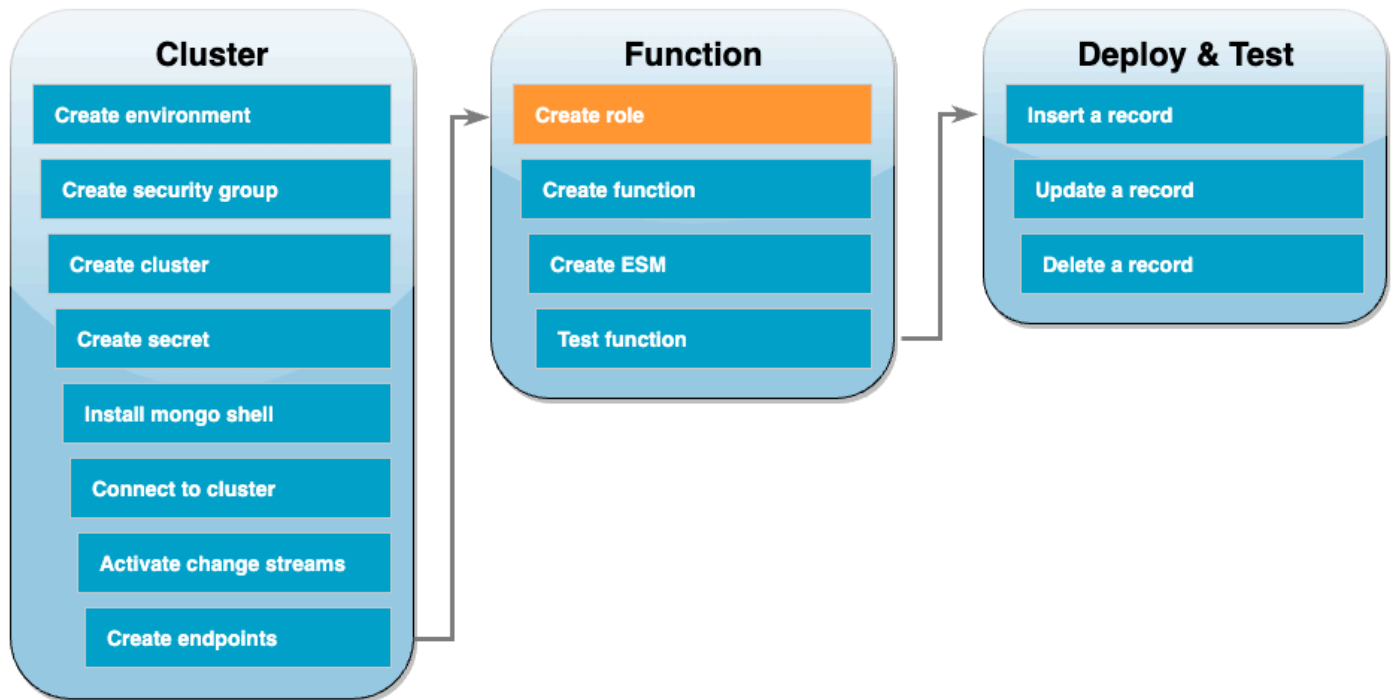
### Cómo crear puntos de conexión de VPC de interfaz

1. Abra la [consola de VPC](#). En el menú de la izquierda, en Nube privada virtual, seleccione Puntos de conexión.
2. Seleccione Crear punto de conexión. Cree un punto de conexión con la siguiente configuración:
  - En Etiqueta de nombre, ingrese `lambda-default-vpc`.
  - En Categoría de servicios, elija Servicios de AWS.
  - En Servicios, ingrese `lambda` en el cuadro de búsqueda. Elija el servicio con formato `com.amazonaws.<region>.lambda`.
  - En VPC, seleccione su [VPC predeterminada](#).
  - En Subredes, marque las casillas situadas junto a cada zona de disponibilidad. Elija el ID de subred correcto para cada zona de disponibilidad.
  - En Tipo de dirección IP, seleccione IPv4.

- En Grupos de seguridad, elija el grupo de seguridad de VPC predeterminado (nombre del grupo de default) y el grupo de seguridad que creó anteriormente (nombre del grupo de DocDBTutorial).
  - Conserve todas las otras opciones de configuración predeterminadas.
  - Seleccione Crear punto de conexión.
3. Elija nuevamente Crear punto de conexión. Cree un punto de conexión con la siguiente configuración:
- En Etiqueta de nombre, ingrese `secretsmanager-default-vpc`.
  - En Categoría de servicios, elija Servicios de AWS.
  - En Servicios, ingrese `secretsmanager` en el cuadro de búsqueda. Elija el servicio con formato `com.amazonaws.<region>.secretsmanager`.
  - En VPC, seleccione su [VPC predeterminada](#).
  - En Subredes, marque las casillas situadas junto a cada zona de disponibilidad. Elija el ID de subred correcto para cada zona de disponibilidad.
  - En Tipo de dirección IP, seleccione IPv4.
  - En Grupos de seguridad, elija el grupo de seguridad de VPC predeterminado (nombre del grupo de default) y el grupo de seguridad que creó anteriormente (nombre del grupo de DocDBTutorial).
  - Conserve todas las otras opciones de configuración predeterminadas.
  - Seleccione Crear punto de conexión.

Esto completa la parte de configuración del clúster de este tutorial.

## Creación del rol de ejecución



En la siguiente serie de pasos, creará la función de Lambda. Primero, debe crear el rol de ejecución que concederá a su función permiso para acceder al clúster. Para ello, primero creará una política de IAM y luego asociará esta política a un rol de IAM.

Para crear una política de IAM

1. En la consola de IAM, abra la [página Políticas](#), y, a continuación, elija Crear política.
2. Seleccione la pestaña JSON. En la siguiente política, sustituya el ARN del recurso de Secrets Manager en la última línea de la instrucción por el ARN secreto anterior y copie la política en el editor.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LambdaESMNetworkingAccess",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",

```

```

        "ec2:DeleteNetworkInterface",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "kms:Decrypt"
    ],
    "Resource": "*"
},
{
    "Sid": "LambdaDocDBESMAccess",
    "Effect": "Allow",
    "Action": [
        "rds:DescribeDBClusters",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBSubnetGroups"
    ],
    "Resource": "*"
},
{
    "Sid": "LambdaDocDBESMGetSecretValueAccess",
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetSecretValue"
    ],
    "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:DocumentDBSecret"
}
]
}

```

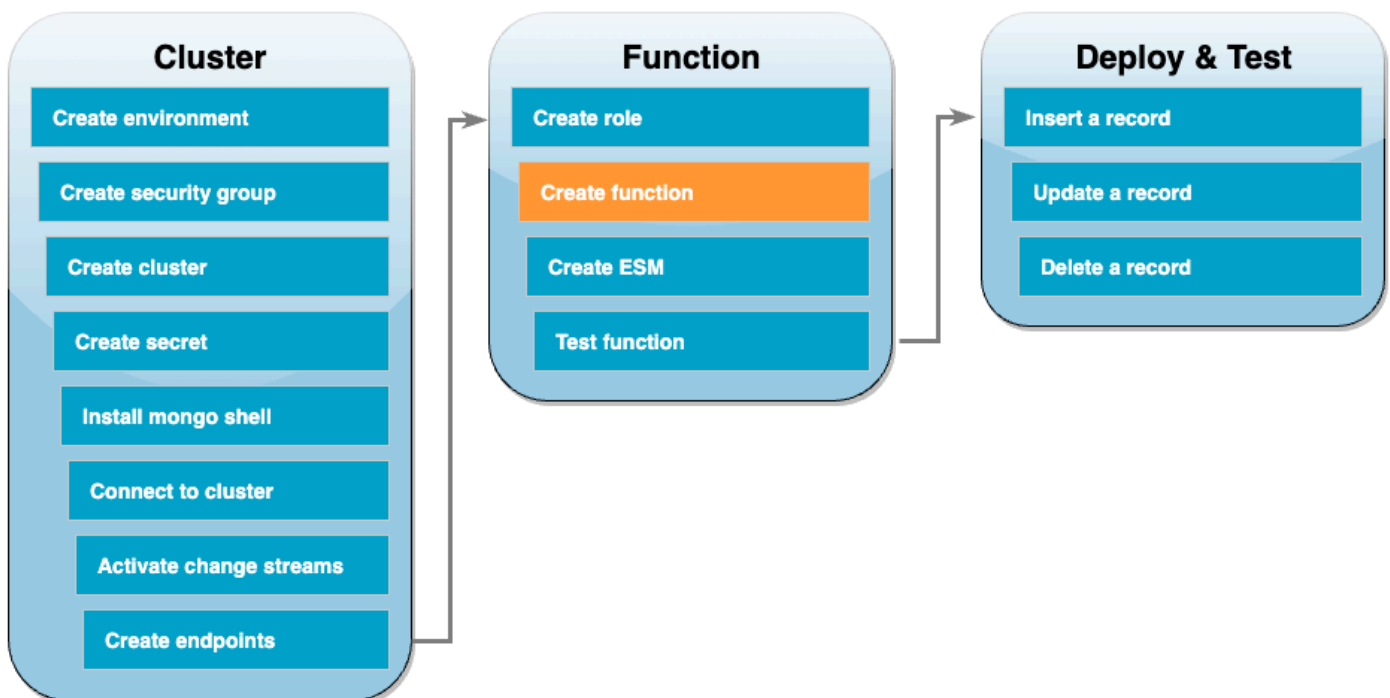
3. Elija Siguiente: Etiquetas y, a continuación, seleccione Siguiente: Revisar.
4. En Name (Nombre), ingrese AWSDocumentDBLambdaPolicy.
5. Elija Crear política.

### Cómo crear el rol de IAM

1. Abra la [página Roles](#) en la consola de IAM y elija Crear rol.
2. En Seleccionar entidad de confianza, elija las siguientes opciones:
  - Tipo de entidad de confianza: servicio de AWS
  - Caso de uso: Lambda
  - Elija Siguiente.

3. En Agregar permisos, elija la política `AWSDocumentDBLambdaPolicy` que acaba de crear y también `AWSLambdaBasicExecutionRole` para conceder a su función permisos para escribir en los Registros de Amazon CloudWatch.
4. Elija Siguiente.
5. En Role name (Nombre del rol), introduzca `AWSDocumentDBLambdaExecutionRole`.
6. Elija Create role (Crear rol).

## Creación de la función de Lambda



El siguiente código de ejemplo recibe un evento de DocumentDB como entrada y procesa el mensaje que contiene.

Go

SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).



## Consumo de un evento de Amazon DocumentDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package main

import (
    "context"
    "encoding/json"
    "fmt"

    "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
    Events []Record `json:"events"`
}

type Record struct {
    Event struct {
        OperationType string `json:"operationType"`
        NS             struct {
            DB string `json:"db"`
            Coll string `json:"coll"`
        } `json:"ns"`
        FullDocument interface{} `json:"fullDocument"`
    } `json:"event"`
}

func main() {
    lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
    fmt.Println("Loading function")
    for _, record := range event.Events {
        logDocumentDBEvent(record)
    }

    return "OK", nil
}

func logDocumentDBEvent(record Record) {
```

```

fmt.Printf("Operation type: %s\n", record.Event.OperationType)
fmt.Printf("db: %s\n", record.Event.NS.DB)
fmt.Printf("collection: %s\n", record.Event.NS.Coll)
docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", " ")
fmt.Printf("Full document: %s\n", string(docBytes))
}

```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de Amazon DocumentDB con Lambda mediante JavaScript.

```

console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};

```

Consumo de un evento de Amazon DocumentDB con Lambda mediante TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de Amazon DocumentDB con Lambda mediante PHP.

```
<?php

require __DIR__.'./vendor/autoload.php';

use Bref\Context\Context;
```

```
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {
        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
"\n";
    }
}

return new DocumentDBEventHandler();
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de Amazon DocumentDB con Lambda mediante Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Consumo de un evento de Amazon DocumentDB con Lambda mediante Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Consumo de un evento de Amazon DocumentDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{"
```

```
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
    Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}
```

Para crear la función de Lambda

1. Copie el código de muestra en un archivo con el nombre `index.js`.
2. Cree un paquete de despliegue con el siguiente comando.

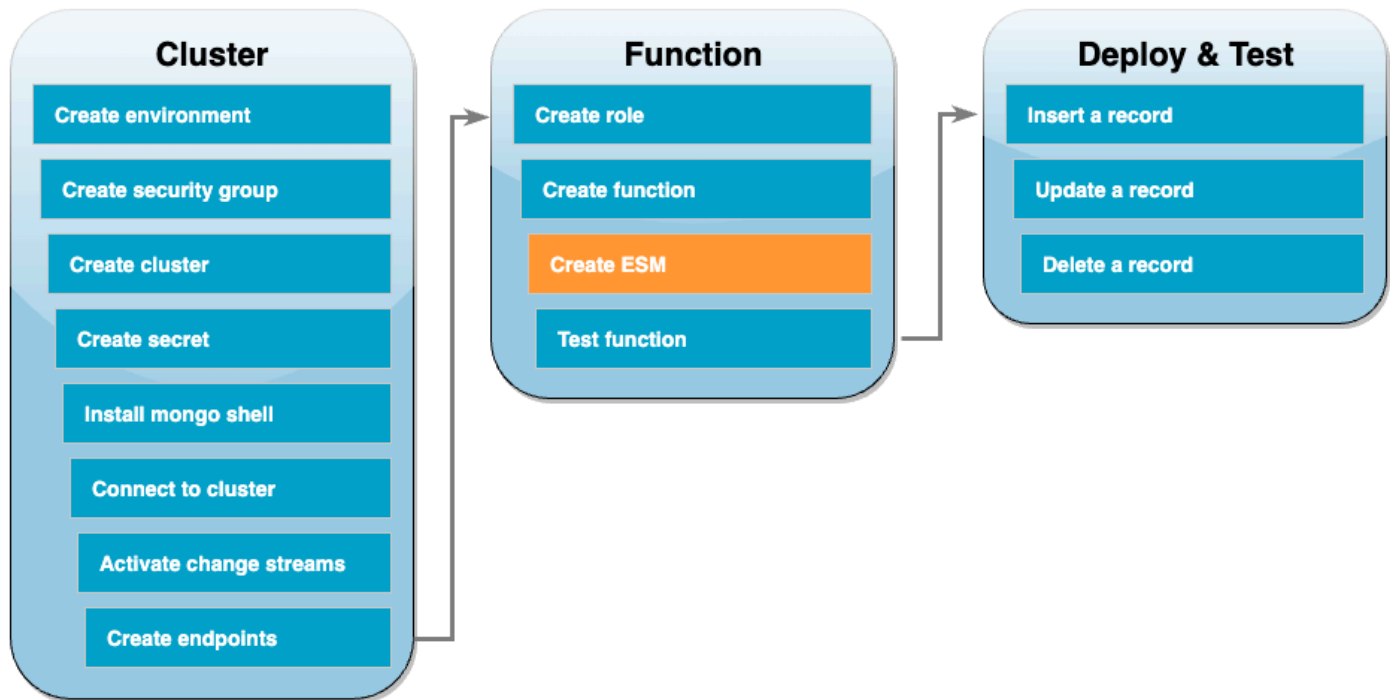
```
zip function.zip index.js
```

3. Use el siguiente comando de la CLI para crear la función. Sustituya `us-east-1` por la región y `123456789012` por su ID de cuenta.

```
aws lambda create-function --function-name ProcessDocumentDBRecords \  
  --zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \  
  --region us-east-1 \  
  --role arn:aws:iam::123456789012:role/AWSDocumentDBLambdaExecutionRole
```



## Crear la asignación de orígenes de eventos de Lambda



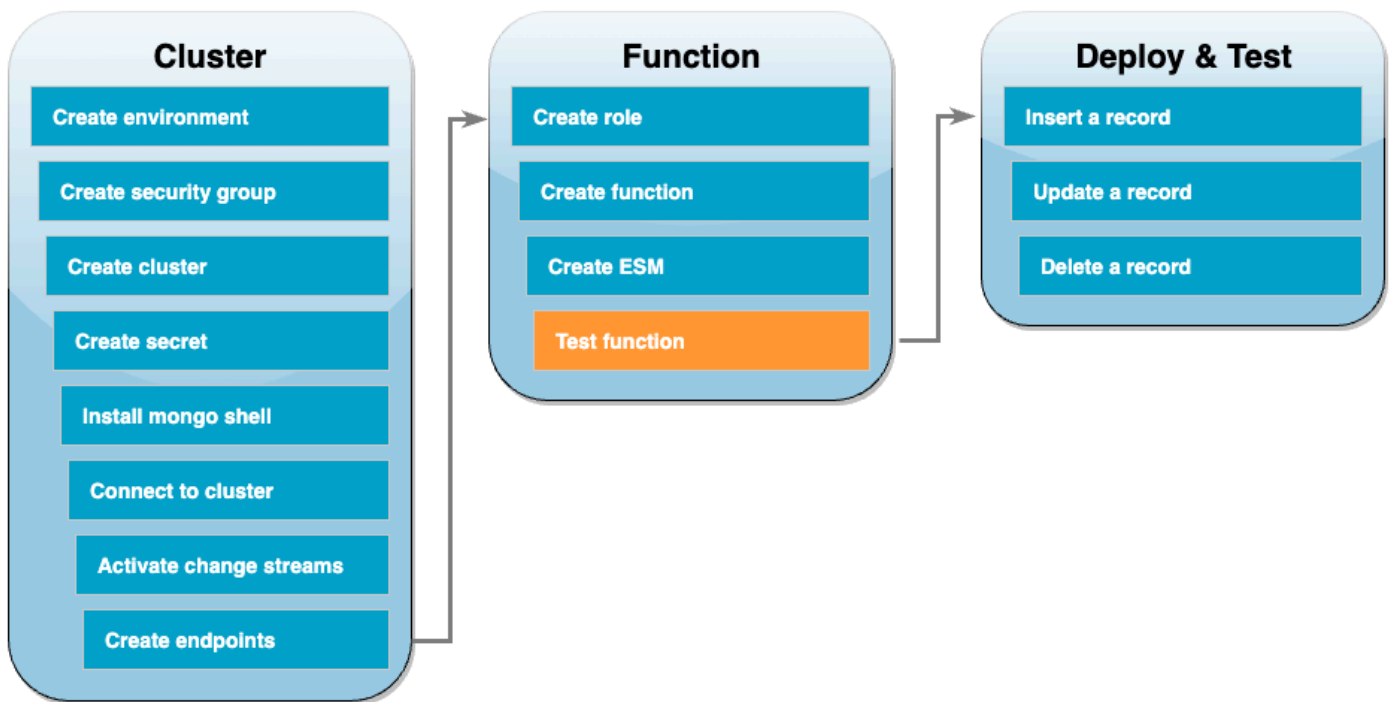
Cree la asignación de orígenes de eventos que asocie su flujo de cambios de DocumentDB a la función de Lambda. Una vez creada esta asignación de orígenes de eventos, AWS Lambda comienza a sondear el flujo.

Para crear la asignación de orígenes de eventos

1. Abra la [página Funciones](#) en la consola de Lambda.
2. Elija la función ProcessDocumentDBRecords que creó anteriormente.
3. Elija la pestaña Configuración y, a continuación, elija Desencadenadores en el menú de la izquierda.
4. Elija Add trigger (Añadir disparador).
5. En Configuración del desencadenador, para el origen, seleccione DocumentDB.
6. Cree la asignación de orígenes de eventos con la siguiente configuración:
  - Clúster de DocumentDB: elija el clúster que creó anteriormente.
  - Nombre de base de datos: docdbdemo
  - Nombre de la colección: productos
  - Batch size: 1

- Posición inicial: más reciente
  - Autenticación: BASIC\_AUTH
  - Clave de Secrets Manager: elija el DocumentDBSecret que acaba de crear.
  - Ventana del lote: 1
  - Configuración completa del documento: UpdateLookup
7. Elija Añadir. Crear la asignación de orígenes de eventos puede tardar unos minutos.

## Probar la función: invocación manual



Para comprobar que ha creado correctamente la función y la asignación de orígenes de eventos, invoque la función con el comando `invoke`. Para ello, primero copie el siguiente evento JSON en un archivo llamado `input.txt`:

```

{
  "eventSourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:canaryclusterb2a659a2-
qo5tcmqkc103",
  "events": [
    {
      "event": {
        "_id": {

```

```
    "_data": "0163eeb6e7000000090100000009000041e1"
  },
  "clusterTime": {
    "$timestamp": {
      "t": 1676588775,
      "i": 9
    }
  },
  "documentKey": {
    "_id": {
      "$oid": "63eeb6e7d418cd98afb1c1d7"
    }
  },
  "fullDocument": {
    "_id": {
      "$oid": "63eeb6e7d418cd98afb1c1d7"
    },
    "anyField": "sampleValue"
  },
  "ns": {
    "db": "docdbdemo",
    "coll": "products"
  },
  "operationType": "insert"
}
],
"eventSource": "aws:docdb"
}
```

Luego use el siguiente comando para invocar la función con este evento:

```
aws lambda invoke --function-name ProcessDocumentDBRecords \  
  --cli-binary-format raw-in-base64-out \  
  --region us-east-1 \  
  --payload file://input.txt out.txt
```

Debería ver una respuesta como la siguiente:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"
```

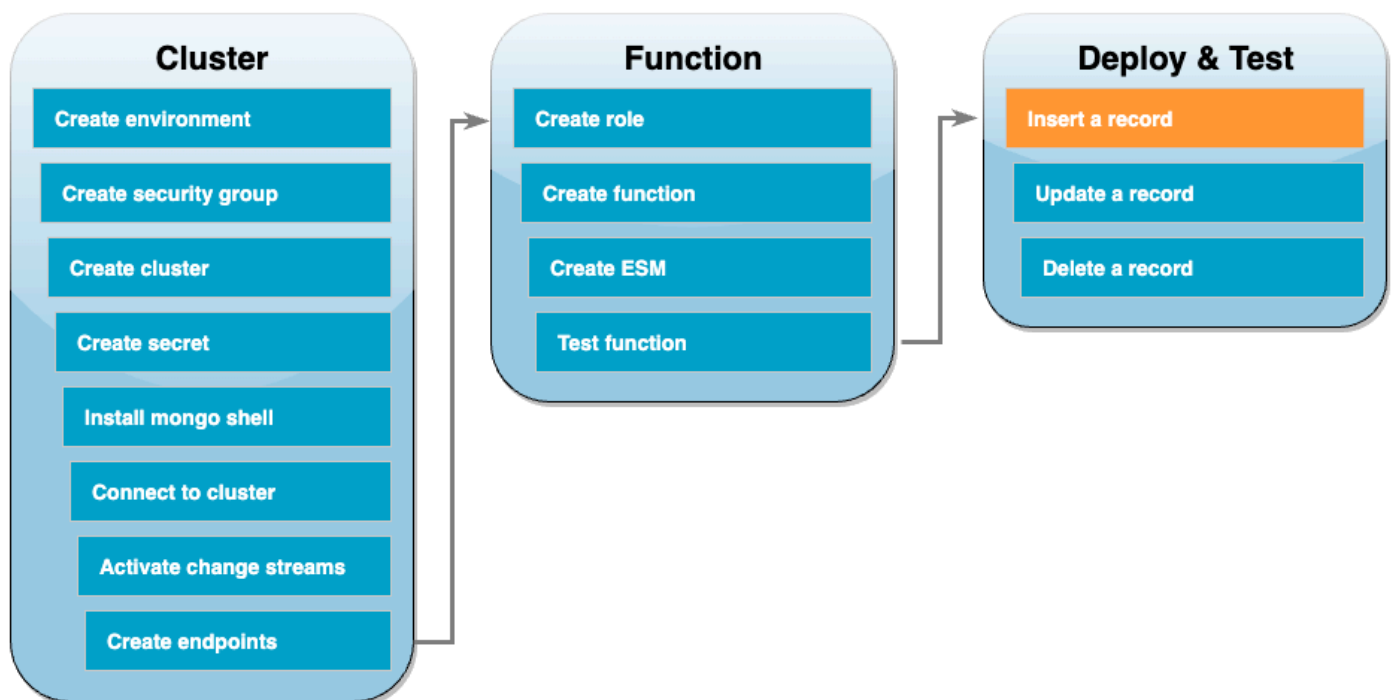
```
}
```

Puede comprobar que su función procesó correctamente el evento al consultar los Registros de CloudWatch.

Cómo verificar la invocación manual mediante los Registros de CloudWatch

1. Abra la [página Funciones](#) en la consola de Lambda.
2. Elija la pestaña Supervisión y luego elija Ver Registros de CloudWatch. Esto lo lleva al grupo de registro específico asociado a su función en la consola de CloudWatch.
3. Elija el flujo de registros más reciente. Dentro de los mensajes de registro, debería ver el JSON del evento.

Probar la función: insertar un registro



Para probar su configuración integral, interactúe directamente con la base de datos de DocumentDB. En la siguiente serie de pasos, insertará un registro, lo actualizará y, a continuación, lo eliminará.

Para insertar un registro

1. [Vuelva a conectarse al clúster de DocumentDB](#) en su entorno de Cloud9.

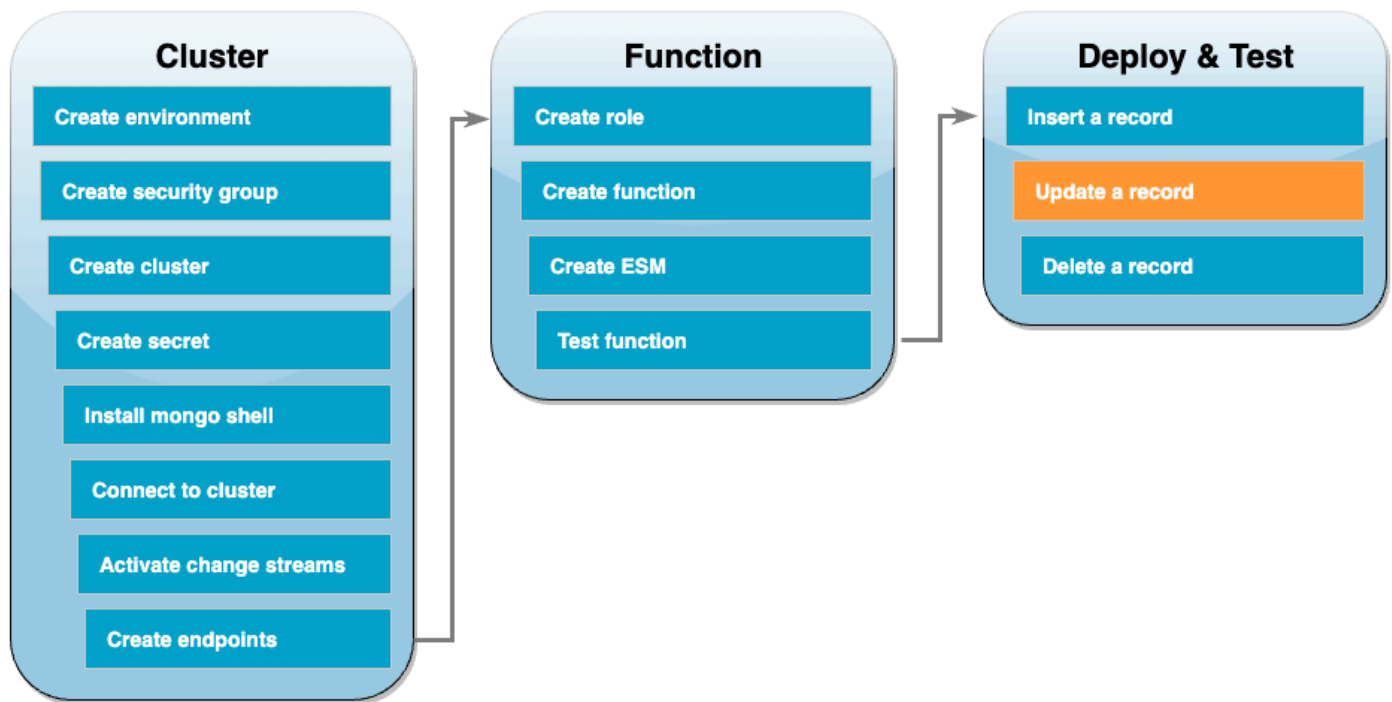
- Use este comando para asegurarse de que está usando la base de datos docdbdemo:

```
use docdbdemo
```

- Inserte un registro en la colección products de la base de datos docdbdemo:

```
db.products.insert({"name":"Pencil", "price": 1.00})
```

## Probar la función: actualizar un registro

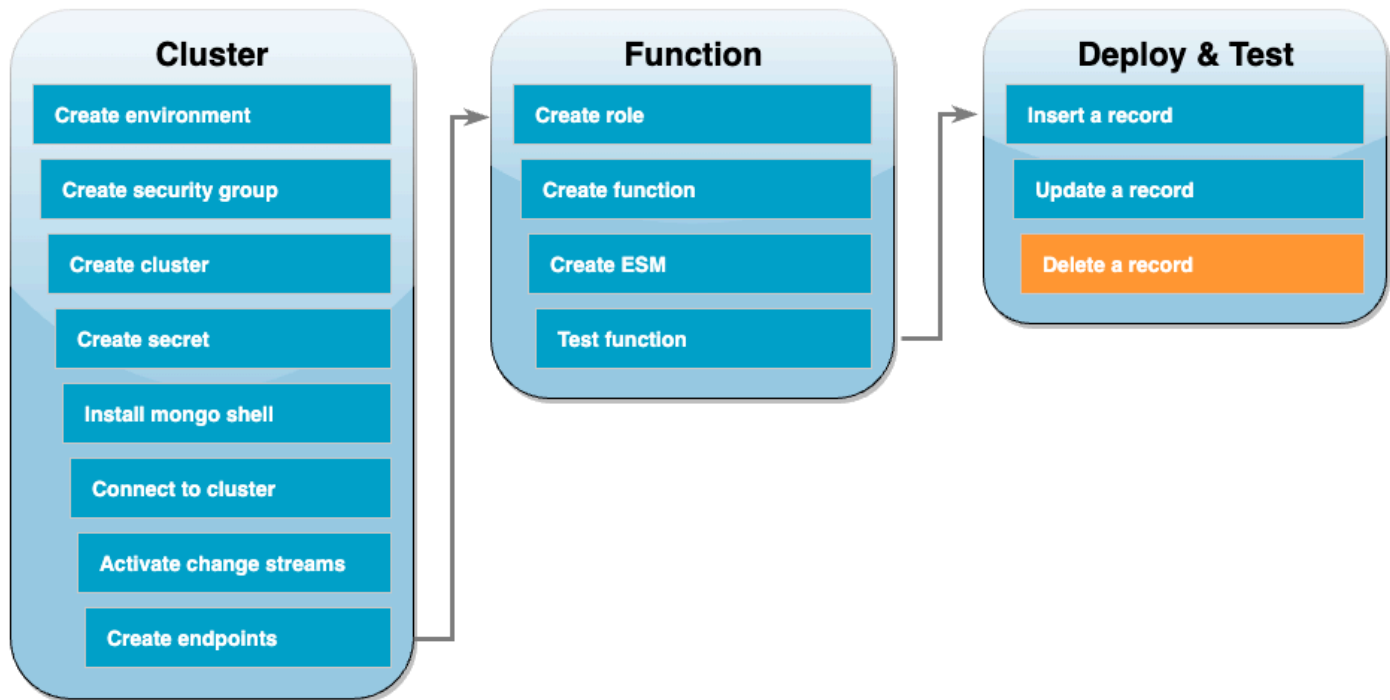


A continuación, actualice el registro que acaba de insertar con el siguiente comando:

```
db.products.update(  
  { "name": "Pencil" },  
  { $set: { "price": 0.50 } }  
)
```

Compruebe que su función procesó correctamente este evento al consultar los Registros de CloudWatch.

## Probar la función: eliminar un registro



Por último, elimine el registro que acaba de actualizar con el siguiente comando:

```
db.products.remove( { "name": "Pencil" } )
```

Compruebe que su función procesó correctamente este evento al consultar los Registros de CloudWatch.

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete(Eliminar).

## Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

## Para eliminar los puntos de conexión de VPC

1. Abra la [consola de VPC](#). En el menú de la izquierda, en Nube privada virtual, seleccione Puntos de conexión.
2. Seleccione los puntos de conexión que ha creado.
3. Elija Acciones, Eliminar puntos de conexión de VPC.
4. Introduzca **delete** en el campo de entrada de texto.
5. Elija Eliminar.

## Para eliminar el clúster de Amazon DocumentDB

1. Abra la [consola de DocumentDB](#).
2. Elija el clúster de DocumentDB que creó para este tutorial y desactive la protección contra la eliminación.
3. En la página principal Clústeres, vuelva a elegir el clúster de DocumentDB.
4. Elija Acciones, Eliminar.
5. En Crear instantánea final del clúster, seleccione No.
6. Introduzca **delete** en el campo de entrada de texto.
7. Elija Eliminar.

## Para eliminar el secreto en Secrets Manager

1. Abra la [consola de Secrets Manager](#).
2. Elija el secreto que ha creado para este tutorial.
3. Elija Acciones, Eliminar secreto.
4. Elija Schedule deletion.

## Para eliminar el grupo de seguridad de Amazon EC2

1. Abra la [consola de EC2](#). En Red y seguridad, elija Grupos de seguridad.
2. Seleccione el grupo de seguridad que ha creado para este tutorial.
3. Elija Acciones, Eliminar grupos de seguridad.
4. Elija Eliminar.

## Para eliminar el entorno de Cloud9

1. Abra la [consola de Cloud9](#).
2. Seleccione el entorno que creó para este tutorial.
3. Elija Eliminar.
4. Introduzca **delete** en el campo de entrada de texto.
5. Elija Eliminar.



# Uso de AWS Lambda con Amazon DynamoDB

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

Puede utilizar una función AWS Lambda para procesar los registros de un [flujo de Amazon DynamoDB](#). Con DynamoDB Streams, puede activar una función de Lambda para realizar trabajo adicional cada vez que se actualice una tabla de DynamoDB.

## Temas

- [Sondeo y procesamiento por lotes de flujos](#)
- [Posiciones iniciales de flujos y sondeo](#)
- [Lectores simultáneos de una partición en DynamoDB Streams](#)
- [Evento de ejemplo](#)
- [Proceso de registros de DynamoDB con Lambda](#)
- [Configuración de la respuesta por lotes parcial con DynamoDB y Lambda](#)
- [Conservación de registros descartados para un origen de eventos de DynamoDB en Lambda](#)
- [Supervisión con Registros de CloudWatch](#)
- [Implementación del procesamiento con estado del flujo DynamoDB en Lambda](#)
- [Parámetros de Lambda para las asignaciones de orígenes de eventos de Amazon DynamoDB](#)
- [Uso del filtrado de eventos con una fuente de eventos de DynamoDB](#)
- [Tutorial: Uso de AWS Lambda con Amazon DynamoDB Streams](#)

## Sondeo y procesamiento por lotes de flujos

Lambda sondea las particiones del DynamoDB Stream y busca registros 4 veces por segundo. Cuando hay registros disponibles, Lambda invoca la función y espera el resultado. Si el procesamiento se realiza correctamente, Lambda reanuda el sondeo hasta que recibe más registros.

De forma predeterminada, Lambda invoca su función tan pronto como los registros estén disponibles. Si el lote que Lambda lee del origen de eventos solo tiene un registro, Lambda envía solo un registro

a la función. Para evitar invocar la función con un número de registros pequeño, puede indicar al origen de eventos que almacene en búfer registros durante hasta 5 minutos configurando un plazo de procesamiento por lotes. Antes de invocar la función, Lambda continúa leyendo los registros del origen de eventos hasta que haya recopilado un lote completo, venza el plazo de procesamiento por lotes o el lote alcance el límite de carga de 6 MB. Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

#### Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

Lambda no espera a que se complete una [extensión](#) configurada antes de enviar el siguiente lote para su procesamiento. En otras palabras, las extensiones pueden seguir ejecutándose mientras Lambda procesa el siguiente lote de registros. Esto puede provocar problemas de limitación si infringe alguno de los ajustes o límites de [simultaneidad](#) de la cuenta. Para detectar si se trata de un posible problema, supervise sus funciones y compruebe si ve [métricas de simultaneidad](#) más elevadas de lo esperado para la asignación de orígenes de eventos. Debido a los tiempos cortos entre invocaciones, Lambda puede informar brevemente un uso de simultaneidad superior al número de particiones. Esto puede ser cierto incluso para las funciones de Lambda sin extensiones.

Ajuste la configuración de [ParallelizationFactor](#) para procesar una partición de un flujo de DynamoDB con más de una invocación de Lambda simultáneamente. Puede especificar el número de lotes simultáneos que Lambda sondea desde una partición a través de un factor de paralelización de 1 (predeterminado) a 10. Por ejemplo, cuando establece `ParallelizationFactor` en 2, puede tener un máximo de 200 invocaciones de Lambda simultáneas para procesar 100 particiones de flujos de DynamoDB (aunque, en la práctica, es posible que observe diferentes valores para la métrica `ConcurrentExecutions`). Esto ayuda a escalar verticalmente el rendimiento de procesamiento cuando el volumen de datos es volátil y el `IteratorAge` es alto. Si aumenta el número de lotes simultáneos por partición, Lambda sigue garantizando el procesamiento en orden del nivel del elemento (partición y clave de clasificación).

## Posiciones iniciales de flujos y sondeo

Tenga en cuenta que el sondeo de flujos durante la creación y las actualizaciones de la asignación de orígenes de eventos es, en última instancia, coherente.

- Durante la creación de la asignación de orígenes de eventos, es posible que se demore varios minutos en iniciar el sondeo de los eventos del flujo.
- Durante las actualizaciones de la asignación de orígenes de eventos, es posible que se demore varios minutos en detener y reiniciar el sondeo de los eventos del flujo.

Este comportamiento significa que, si especifica LATEST como posición inicial del flujo, la asignación de orígenes de eventos podría omitir eventos durante la creación o las actualizaciones. Para garantizar que no se pierda ningún evento, especifique la posición inicial del flujo como TRIM\_HORIZON.

## Lectores simultáneos de una partición en DynamoDB Streams

En el caso de las tablas de una sola región que no sean tablas globales, puede diseñar hasta dos funciones de Lambda para leer desde la misma partición de DynamoDB Streams al mismo tiempo. Si excede este límite, puede producirse una limitación controlada de las solicitudes. En el caso de las tablas globales, le recomendamos que limite el número de funciones simultáneas a uno para evitar la limitación de solicitudes.

## Evento de ejemplo

### Example

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        }
      },
      "NewImage": {
        "Message": {
```

```
        "S": "New item!"
    },
    "Id": {
        "N": "101"
    }
},
"StreamViewType": "NEW_AND_OLD_IMAGES",
"SequenceNumber": "111",
"SizeBytes": 26
},
"awsRegion": "us-west-2",
"eventName": "INSERT",
"eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
"eventSource": "aws:dynamodb"
},
{
    "eventID": "2",
    "eventVersion": "1.0",
    "dynamodb": {
        "OldImage": {
            "Message": {
                "S": "New item!"
            },
            "Id": {
                "N": "101"
            }
        },
        "SequenceNumber": "222",
        "Keys": {
            "Id": {
                "N": "101"
            }
        }
    },
    "SizeBytes": 59,
    "NewImage": {
        "Message": {
            "S": "This item has changed"
        },
        "Id": {
            "N": "101"
        }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES"
```

```

    },
    "awsRegion": "us-west-2",
    "eventName": "MODIFY",
    "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
    "eventSource": "aws:dynamodb"
  }
]}

```

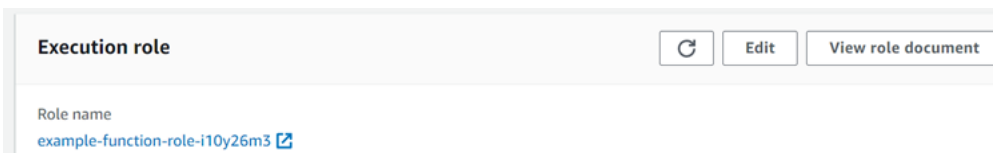
## Proceso de registros de DynamoDB con Lambda

Cree una asignación de orígenes de eventos para indicar a Lambda que envíe registros desde un flujo a una función de Lambda. Puede crear varias asignaciones de orígenes de eventos para procesar los mismos datos con distintas funciones de Lambda o para procesar elementos de varios flujos con una sola función.

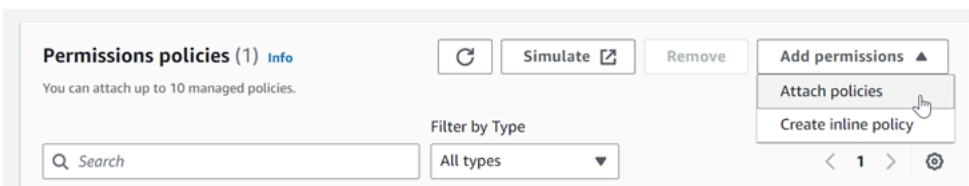
Para configurar la función para que lea desde el flujo de DynamoDB, adjunte la política administrada de AWS [AWSLambdaDynamoDBExecutionRole](#) al rol de ejecución y, a continuación, cree un desencadenador de DynamoDB.

Cómo agregar permisos y crear un desencadenador

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija la pestaña Configuración y, a continuación, elija Permisos.
4. En Nombre del rol, elija el enlace al rol de ejecución. Este enlace abre el rol en la consola de IAM.



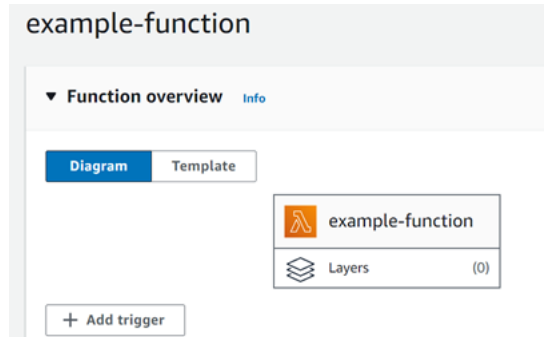
5. Elija Agregar permisos y luego Adjuntar políticas.



6. En el campo de búsqueda, escriba `AWSLambdaDynamoDBExecutionRole`. Agregue esta política al rol de ejecución. Se trata de una política administrada por AWS que contiene los

permisos que la función necesita para leer desde un flujo de DynamoDB. Para obtener más información acerca de esta política, consulte [AWSLambdaDynamoDBExecutionRole](#) en la Referencia de políticas administradas de AWS.

7. Regrese a la función en la consola de Lambda. En Descripción general de la función, elija Agregar desencadenador.



8. Elija un tipo de desencadenador.
9. Configure las opciones requeridas y luego elija Add (Agregar).

Lambda admite las siguientes opciones para los orígenes de eventos de DynamoDB:

#### Opciones de origen de eventos

- Tabla DynamoDB: la tabla de DynamoDB de la que leer registros.
- Tamaño del lote: número de registros que se enviarán a la función en cada lote, hasta 10 000. Lambda pasa todos los registros del lote a la función en una sola llamada, siempre y cuando el tamaño total de los eventos no exceda el [límite de carga](#) para la invocación síncrona (6 MB).
- Ventana de lote: especifique la cantidad de tiempo máxima para recopilar registros antes de invocar la función, en segundos.
- Posición inicial: procesar solo los registros nuevos o todos los registros existentes.
  - Más recientes: procesar los registros nuevos que se agreguen al flujo principal.
  - Horizonte de supresión: procesar todos los registros del flujo.

Tras procesar cualquier registro existente, la función es alcanzada y continúa procesando registros nuevos.

- Destino en caso de error: una cola de SQS estándar o un tema de SNS estándar para los registros que no se puedan procesar. Cuando Lambda descarta un lote de registros demasiado antiguo o que ha agotado todos los reintentos, Lambda envía detalles sobre el lote a la cola o al tema.

- **Número de reintentos:** número máximo de reintentos que Lambda realiza cuando la función devuelve un error. Esto no se aplica a errores de servicio o limitaciones controladas en los que el lote no alcanzó la función.
- **Edad máxima de registro:** antigüedad máxima de un registro que Lambda envía a su función.
- **División del lote en caso de error:** cuando la función devuelve un error, divide el lote en dos antes de volver a intentarlo. La configuración de tamaño de lote original permanece sin cambios.
- **Lotes simultáneos por partición:** procese simultáneamente varios lotes desde la misma partición.
- **Habilitado:** establézcalo en verdadero para habilitar la asignación de orígenes de eventos. Establézcalo en falso para detener el procesamiento de registros. Lambda toma nota del último registro procesado y sigue procesando desde ese punto cuando se habilita de nuevo el mapeo.

#### Note

No se cobran las llamadas a la API `GetRecords` invocadas por Lambda como parte de los desencadenadores de DynamoDB.

Para administrar la configuración de origen de evento más tarde, elija el desencadenador en el diseñador.

## Configuración de la respuesta por lotes parcial con DynamoDB y Lambda

Al consumir y procesar datos de streaming desde un origen de eventos, de forma predeterminada, Lambda comprueba hasta el número de secuencia más alto de un lote solo cuando el lote se ha completado con éxito. Lambda trata todos los demás resultados como un error completo y vuelve a intentar procesar el lote hasta el límite de reintentos. Para permitir éxitos parciales al procesar lotes de una secuencia, active `ReportBatchItemFailures`. Permitir éxitos parciales puede ayudar a reducir el número de reintentos en un registro, aunque no impide por completo la posibilidad de reintentos en un registro exitoso.

Para activar `ReportBatchItemFailures`, incluya el valor enumerado

**`ReportBatchItemFailures`** en la lista [FunctionResponseTypes](#). Esta lista indica qué tipos de respuesta están habilitados para su función. Puede configurar esta lista al [crear](#) o [actualizar](#) una asignación de orígenes de eventos.

## Sintaxis del informe

Al configurar los informes sobre errores de elementos por lotes, la clase `StreamsEventResponse` se devuelve con una lista de errores de elementos de lote. Puede utilizar un objeto `StreamsEventResponse` para devolver el número de secuencia del primer registro fallido del lote. También puede crear su propia clase personalizada usando la sintaxis de respuesta correcta. La siguiente estructura JSON muestra la sintaxis de respuesta requerida:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": "<SequenceNumber>"
    }
  ]
}
```

### Note

Si la matriz `batchItemFailures` contiene varios elementos, Lambda usa el registro con el número de secuencia más bajo como punto de control. Luego Lambda vuelve a probar todos los registros a partir de ese punto de control.

## Condiciones de éxito y fracaso

Lambda trata un lote como un éxito completo si devuelve cualquiera de los siguientes elementos:

- Una lista `batchItemFailure` vacía
- Una lista `batchItemFailure` nula
- Una `EventResponse` vacía
- Un `EventResponse` nulo

Lambda trata un lote como un error completo si devuelve cualquiera de los siguientes elementos:

- Una cadena `itemIdentifier` vacía
- Una `itemIdentifier` nula
- Un `itemIdentifier` con un mal nombre de clave



Lambda reintentos fallidos basados en su estrategia de reintento.

## Bisecar un lote

Si su invocación falla y `BisectBatchOnFunctionError` está activada, el lote se divide en bisectos independientemente de su configuración `ReportBatchItemFailures`.

Cuando se recibe una respuesta de éxito parcial de lote y se activan tanto `BisectBatchOnFunctionError` como `ReportBatchItemFailures`, el lote se divide en el número de secuencia devuelto y Lambda vuelve a intentar solo los registros restantes.

Estos son algunos ejemplos de código de función que devuelven la lista de IDs de mensajes fallidos del lote:

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
```

```
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();


        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

## Go

## SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
    ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
    BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
(*BatchResult, error) {
    var batchItemFailures []BatchItemFailure
    curRecordSequenceNumber := ""

    for _, record := range event.Records {
        // Process your record
        curRecordSequenceNumber = record.Change.SequenceNumber
    }

    if curRecordSequenceNumber != "" {
        batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
curRecordSequenceNumber})
    }
}
```

```
}

batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
}

return &batchResult, nil
}

func main() {
  lambda.Start(HandleRequest)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
  Serializable> {
```

```
@Override
public StreamsEventResponse handleRequest(DynamodbEvent input, Context
context) {

    List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
    ArrayList<>();
    String curRecordSequenceNumber = "";

    for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
    input.getRecords()) {
        try {
            //Process your record
            StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
            curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

        } catch (Exception e) {
            /* Since we are working with streams, we can return the failed
            item immediately.
            Lambda will immediately begin to retry processing from this
            failed item onwards. */
            batchItemFailures.add(new
            StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
            return new StreamsEventResponse(batchItemFailures);
        }
    }

    return new StreamsEventResponse();
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

## Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;
```

```
    if (curRecordSequenceNumber) {
        batchItemFailures.push({
            itemIdentifier: curRecordSequenceNumber,
        });
    }
}

return { batchItemFailures: batchItemFailures };
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante PHP.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
```

```
public function __construct(StderrLogger $logger)
{
    $this->logger = $logger;
}

/**
 * @throws JsonException
 * @throws \Bref\Event\InvalidLambdaEvent
 */
public function handle(mixed $event, Context $context): array
{
    $dynamoDbEvent = new DynamoDbEvent($event);
    $this->logger->info("Processing records");

    $records = $dynamoDbEvent->getRecords();
    $failedRecords = [];
    foreach ($records as $record) {
        try {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data
        } catch (Exception $e) {
            $this->logger->error($e->getMessage());
            // failed processing the record
            $failedRecords[] = $record->getSequenceNumber();
        }
    }
    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords records");

    // change format for the response
    $failures = array_map(
        fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
        $failedRecords
    );

    return [
        'batchItemFailures' => $failures
    ];
}

$logger = new StderrLogger();
return new Handler($logger);
```



## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  records = event["Records"]
  cur_record_sequence_number = ""

  records.each do |record|
    begin
      # Process your record
      cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
cur_record_sequence_number}]}
    end
  end

  {"batchItemFailures" => []}
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
    Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
    }
}
```

```

    return Ok(response);
}

for record in records {
    tracing::info!("EventId: {}", record.event_id);

    // Couldn't find a sequence number
    if record.change.sequence_number.is_none() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: Some("").to_string(),
        });
        return Ok(response);
    }

    // Process your record here...
    if process_record(record).is_err() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: record.change.sequence_number.clone(),
        });
        /* Since we are working with streams, we can return the failed item
immediately.
        Lambda will immediately begin to retry processing from this failed
item onwards. */
        return Ok(response);
    }
}

tracing::info!("Successfully processed {} record(s)", records.len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

```
}
```

## Conservación de registros descartados para un origen de eventos de DynamoDB en Lambda

La gestión de errores en las asignaciones de orígenes de eventos de DynamoDB depende de si el error se produce antes de que se invoque la función o durante la invocación de la función:

- Antes de la invocación: si una asignación de orígenes de eventos de Lambda no puede invocar la función debido a una limitación u otros problemas, lo vuelve a intentar hasta que los registros caduquen o superen la antigüedad máxima configurada en la asignación de orígenes de eventos ([MaximumRecordAgeInSeconds](#)).
- Durante invocación: [si se invoca la función pero devuelve un error, Lambda vuelve a intentarlo hasta que los registros caduquen, superen la antigüedad máxima \(MaximumRecordAgeInSeconds\) o alcancen la cuota de reintento configurada \(MaximumRetryAttempts\)](#). En el caso de errores de función, también puede configurar [BisectBatchOnFunctionError](#), que divide un lote fallido en dos lotes más pequeños, aislando registros fallidos y evitando tiempos de espera. La división de lotes no consume la cuota de reintentos.

Si las medidas de administración de errores fallan, Lambda descarta los registros y continúa procesando lotes del flujo. Con la configuración predeterminada, esto significa que un registro incorrecto puede bloquear el procesamiento en la partición afectada durante un máximo de un día. Para evitar esto, configure el mapeo de fuente de eventos de su función con un número razonable de reintentos y una antigüedad máxima de registro que se ajuste a su caso de uso.

## Configuración de destinos para invocaciones fallidas

Para retener los registros de las invocaciones de asignación de orígenes de eventos fallidos, agregue un destino a la asignación de orígenes de eventos de su función. Cada registro enviado al destino es un documento JSON con metadatos sobre la invocación fallida. Puede configurar cualquier tema de Amazon SNS o cualquier cola de Amazon SQS como destino. Su rol de ejecución debe tener permisos para el destino:

- Para destinos de SQS: [sqs:SendMessage](#)
- Para destinos SNS: [sns:Publish](#)

Para configurar un destino en caso de error mediante la consola, siga estos pasos:

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En Descripción general de la función, elija Agregar destino.
4. En Origen, elija Invocación de asignación de orígenes de eventos.
5. Para la Asignación de orígenes de eventos, elija un origen de eventos que esté configurado para esta función.
6. En Condición, seleccione En caso de error. Para las invocaciones de asignación de orígenes de eventos, esta es la única condición aceptada.
7. En Tipo de destino, elija el tipo de destino al que Lambda envía los registros de invocación.
8. En Destino, elija un recurso.
9. Seleccione Guardar.

También puede configurar un destino en caso de error mediante la AWS Command Line Interface (AWS CLI). Por ejemplo, el siguiente comando [create-event-source-mapping](#) agrega una asignación de orígenes de eventos con un destino de SQS en caso de error a MyFunction:

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2024-06-10T19:26:16.525 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```

El siguiente comando [update-event-source-mapping](#) actualiza una asignación de orígenes de eventos para enviar registros de invocación fallida a un destino de SNS después de dos intentos de reintento, o si los registros tienen más de una hora de antigüedad.

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--maximum-retry-attempts 2 \  
--maximum-record-age-in-seconds 3600 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-  
east-1:123456789012:dest-topic"}}'
```

La configuración actualizada se aplica de forma asincrónica y no se refleja en la salida hasta que se completa el proceso. Utilice el comando [get-event-source-mapping](#) para ver el estado actual.

Para eliminar un destino, introduzca una cadena vacía como argumento del parámetro `destination-config`:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": ""}}'
```

El siguiente ejemplo muestra un registro de invocación para un flujo de DynamoDB Stream.

Example Registro de invocación

```
{  
  "requestContext": {  
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",  
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
    "condition": "RetryAttemptsExhausted",  
    "approximateInvokeCount": 1  
  },  
  "responseContext": {  
    "statusCode": 200,  
    "executedVersion": "$LATEST",  
    "functionError": "Unhandled"  
  },  
  "version": "1.0",  
  "timestamp": "2019-11-14T00:13:49.717Z",  
  "DDBStreamBatchInfo": {  
    "shardId": "shardId-00000001573689847184-864758bb",  
    "startSequenceNumber": "800000000003126276362",  
    "endSequenceNumber": "800000000003126276362",  
    "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",  
    "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",  
    "batchSize": 1,  
    "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/  
stream/2019-11-14T00:04:06.388"  
  }  
}
```

Puede utilizar esta información para recuperar los registros afectados del flujo para solucionar problemas. Los registros reales no están incluidos, por lo que debe procesar este registro y recuperarlos del flujo antes de que caduquen y se pierdan.

## Supervisión con Registros de CloudWatch

Lambda emite la métrica `IteratorAge` cuando la función termina de procesar un lote de registros. La métrica indica la antigüedad del último registro del lote cuando acabo el proceso. Si la función está procesando nuevos eventos, puede utilizar la antigüedad del iterador para estimar la latencia entre cuando un registro se añade y cuando la función lo procesa.

Una tendencia ascendente en la antigüedad del iterador puede indicar problemas con la función. Para obtener más información, consulte [Ver las métricas de funciones de Lambda](#).

## Implementación del procesamiento con estado del flujo DynamoDB en Lambda

Las funciones de Lambda pueden ejecutar aplicaciones de procesamiento de flujo continuo. Una secuencia representa datos ilimitados que fluyen de forma continua a través de su aplicación. Para analizar la información de esta entrada de actualización continua, puede enlazar los registros incluidos mediante una ventana definida en términos de tiempo.

Las ventanas de salto constante son ventanas de tiempo distintas que se abren y cierran a intervalos regulares. De forma predeterminada, las invocaciones de Lambda no tienen estado: no se pueden utilizar para procesar datos en múltiples invocaciones continuas sin una base de datos externa. Sin embargo, con las ventanas de salto constante, puede mantener su estado en todas las invocaciones. Este estado contiene el resultado agregado de los mensajes procesados previamente para la ventana actual. Su estado puede ser un máximo de 1 MB por partición. Si supera ese tamaño, Lambda finaliza la ventana antes de tiempo.

Cada registro de una secuencia pertenece a un periodo específico. Lambda procesará cada registro al menos una vez, pero no garantiza que cada registro se procese solo una vez. En casos excepcionales, como el manejo de errores, es posible que algunos registros se procesen más de una vez. Los registros siempre se procesan en orden la primera vez. Si los registros se procesan más de una vez, es posible que lo hagan de forma desordenada.



## Agregación y procesamiento

Su función administrada por el usuario se invoca tanto para la agregación como para procesar los resultados finales de esa agregación. Lambda agrega todos los registros recibidos en la ventana. Puede recibir estos registros en varios lotes, cada uno como una invocación independiente. Cada invocación recibe un estado. Por lo tanto, al usar las ventanas de salto constante, su respuesta de la función de Lambda debe contener una propiedad de `state`. Si la respuesta no contiene una propiedad de `state`, Lambda considera que esto es una invocación fallida. Para satisfacer esta condición, la función puede devolver un objeto de `TimeWindowEventResponse`, que tiene la siguiente forma JSON:

### Example Valores `TimeWindowEventResponse`

```
{
  "state": {
    "1": 282,
    "2": 715
  },
  "batchItemFailures": []
}
```

#### Note

Para las funciones Java, se recomienda utilizar un `Map<String, String>` para representar el estado.

Al final de la ventana, el indicador `isFinalInvokeForWindow` está configurado en `true` para indicar que este es el estado final y que está listo para su procesamiento. Después del procesamiento, la ventana se completa y su invocación final se completa, y luego se elimina el estado.

Al final de la ventana, Lambda utiliza el procesamiento final para las acciones en los resultados de agregación. Su procesamiento final se invoca sincrónicamente. Después de la invocación exitosa, los puntos de control de la función, el número de secuencia y el procesamiento de flujo continúa. Si la invocación no tiene éxito, su función de Lambda suspende el procesamiento posterior hasta una invocación exitosa.

## Example DynamoDBTimeWindowEvent

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "111",
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "stream-ARN"
    },
    {
      "eventID": "2",
      "eventName": "MODIFY",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
```

```
    "Message":{
      "S":"This item has changed"
    },
    "Id":{
      "N":"101"
    }
  },
  "OldImage":{
    "Message":{
      "S":"New item!"
    },
    "Id":{
      "N":"101"
    }
  },
  "SequenceNumber":"222",
  "SizeBytes":59,
  "StreamViewType":"NEW_AND_OLD_IMAGES"
},
"eventSourceARN":"stream-ARN"
},
{
  "eventID":"3",
  "eventName":"REMOVE",
  "eventVersion":"1.0",
  "eventSource":"aws:dynamodb",
  "awsRegion":"us-east-1",
  "dynamodb":{
    "Keys":{
      "Id":{
        "N":"101"
      }
    }
  },
  "OldImage":{
    "Message":{
      "S":"This item has changed"
    },
    "Id":{
      "N":"101"
    }
  },
  "SequenceNumber":"333",
  "SizeBytes":38,
  "StreamViewType":"NEW_AND_OLD_IMAGES"
```

```

    },
    "eventSourceARN": "stream-ARN"
  }
],
"window": {
  "start": "2020-07-30T17:00:00Z",
  "end": "2020-07-30T17:05:00Z"
},
"state": {
  "1": "state1"
},
"shardId": "shard123456789",
"eventSourceARN": "stream-ARN",
"isFinalInvokeForWindow": false,
"isWindowTerminatedEarly": false
}

```

## Configuración

Puede configurar ventanas de salto constante al crear o actualizar una asignación de orígenes de eventos. Para configurar una ventana de saltos de tamaño constante, especifique la ventana en segundos ([TumblingWindowInSeconds](#)). El siguiente comando de ejemplo AWS Command Line Interface (AWS CLI) crea una asignación de origen de eventos de streaming que tiene una ventana de salto constante de 120 segundos. Se nombra la función de Lambda definida para la agregación y el procesamiento se llama `tumbling-window-example-function`.

```

aws lambda create-event-source-mapping \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525 \
--function-name tumbling-window-example-function \
--starting-position TRIM_HORIZON \
--tumbling-window-in-seconds 120

```

Lambda determina los límites de la ventana de salto constante en función de la hora en que se insertaron los registros en la secuencia. Todos los registros tienen una marca de hora aproximada disponible que Lambda utiliza en las determinaciones de límites.

Las agregaciones de ventanas de saltos constantes no admiten el reendurecimiento. Cuando el fragmento termina, Lambda considera la ventana cerrada y las particiones secundarias comienzan su propia ventana en un estado fresco.

Ventanas de saltos constantes son totalmente compatibles con las directivas de reintento existentes `maxRetryAttempts` y `maxRecordAge`.

Example Handler.py: agregación y procesamiento

La siguiente función de Python muestra cómo agregar y luego procesar su estado final:

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

    #Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
    else:
        print('Aggregate invoke')

    #Check for early terminations
    if event['isWindowTerminatedEarly']:
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['dynamodb']['NewImage']['Id']] = state.get(record['dynamodb']
['NewImage']['Id'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

## Parámetros de Lambda para las asignaciones de orígenes de eventos de Amazon DynamoDB

Todos los tipos de fuente de eventos Lambda comparten las mismas operaciones

[CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Amazon DynamoDB Streams.

Parámetro	Obligatoria	Predeterminado	Notas
BatchSize	N	100	Máximo: 10 000

Parámetro	Obligatoria	Predeterminado	Notas
BisectBatchOnFunctionError	N	false	Ninguno
DestinationConfig	N	N/A	Cola de Amazon SQS estándar o destino de tema de Amazon SNS estándar para registros descartados
Habilitado	N	true	Ninguno
EventSourceArn	Y	N/A	ARN del flujo de datos o un consumidor de flujos
FilterCriteria	N	N/A	<a href="#">Controle qué eventos envía Lambda a la función</a>
FunctionName	Y	N/A	Ninguno
FunctionResponseTypes	N	N/A	Para permitir que la función informe de errores específicos de un lote, incluya el valor ReportBatchItemFailures en FunctionResponseTypes. Para obtener más información, consulte <a href="#">Configuración de la respuesta por lotes parcial con DynamoDB y Lambda</a> .

Parámetro	Obligatoria	Predeterminado	Notas
MaximumBatchingWindowInSeconds	N	0	Ninguno
MaximumRecordAgeInSeconds	N	-1	-1 significa infinito: se vuelven a intentar los registros que han producido error hasta que caduque el registro. El <a href="#">límite de retención de datos del flujo de DynamoDB</a> es de 24 horas.  Mínimo: -1  Máximo: 604 800
MaximumRetryAttempts	N	-1	-1 significa infinito: se vuelven a intentar los registros que han producido error hasta que caduque el registro  Mínimo: 0  Máximo: 10 000
ParallelizationFactor	N	1	Máximo: 10
StartingPosition	Y	N/A	TRIM_HORIZON o LATEST
TumblingWindowInSeconds	N	N/A	Mínimo: 0  Máximo: 900

## Uso del filtrado de eventos con una fuente de eventos de DynamoDB

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo funciona el filtrado de eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para las fuentes de eventos de DynamoDB.

### Temas

- [evento de DynamoDB](#)
- [Filtrar con atributos de tabla](#)
- [Filtrar con expresiones booleanas](#)
- [Uso del operador Exists](#)
- [Formato JSON para filtrado de DynamoDB](#)

### evento de DynamoDB

Supongamos que tiene una tabla de DynamoDB con la clave principal `CustomerName` y los atributos `AccountManager` y `PaymentTerms`. El siguiente es un registro de ejemplo de flujo de la tabla de DynamoDB.

```
{
  "eventID": "1",
  "eventVersion": "1.0",
  "dynamodb": {
    "ApproximateCreationDateTime": "1678831218.0",
    "Keys": {
      "CustomerName": {
        "S": "AnyCompany Industries"
      },
      "NewImage": {
        "AccountManager": {
          "S": "Pat Candella"
        },
        "PaymentTerms": {
          "S": "60 days"
        },
        "CustomerName": {
          "S": "AnyCompany Industries"
        }
      }
    }
  }
}
```



```

        }
      },
      "SequenceNumber": "111",
      "SizeBytes": 26,
      "StreamViewType": "NEW_IMAGE"
    }
  }
}

```

Para filtrar en función de los valores de clave y atributos en la tabla de DynamoDB, utilice la clave dynamodb del registro. En las siguientes secciones, se muestran ejemplos de diferentes tipos de filtros.

### Filtrar con claves de tabla

Supongamos que desea que su función procese únicamente los registros en los que la clave principal CustomerName sea “AnyCompany Industries”. El objeto FilterCriteria sería el siguiente.

```

{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"
    }
  ]
}

```

Para mayor claridad, este es el valor del Pattern del filtro ampliado en JSON no cifrado.

```

{
  "dynamodb": {
    "Keys": {
      "CustomerName": {
        "S": [ "AnyCompany Industries" ]
      }
    }
  }
}

```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany Industries" ] } } } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany Industries" ] } } } }'
```

## Filtrar con atributos de tabla

Con DynamoDB, también puede utilizar las claves `NewImage` y `OldImage` para filtrar por los valores de los atributos. Supongamos que desea filtrar los registros en los que el atributo `AccountManager` de la última imagen de la tabla sea “Pat Candella” o “Shirley Rodriguez”. El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "dynamodb": {
    "NewImage": {
      "AccountManager": {
        "S": [ "Pat Candella", "Shirley Rodriguez" ]
      }
    }
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
"Shirley Rodriguez" ] } } } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"]}]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"]}]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
Candella", "Shirley Rodriguez" ] } } } }'
```

## Filtrar con expresiones booleanas

También puede crear filtros mediante expresiones booleanas AND. Estas expresiones pueden incluir tanto los parámetros de clave como los de atributo de la tabla. Supongamos que desea filtrar los registros en los que el valor `NewImage` de `AccountManager` es “Pat Candella” y el valor `OldImage` es “Terry Whitlock”. El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
```

```

    {
      "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager
\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
    }
  ]
}

```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```

{
  "dynamodb" : {
    "NewImage" : {
      "AccountManager" : {
        "S" : [
          "Pat Candella"
        ]
      }
    }
  },
  "dynamodb": {
    "OldImage": {
      "AccountManager": {
        "S": [
          "Terry Whitlock"
        ]
      }
    }
  }
}

```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```

{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
[ "Terry Whitlock" ] } } } }

```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage \" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } } ]}]'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage \" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } } ]}]'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" : [ "Terry Whitlock" ] } } } }'
```

### Note

El filtrado de eventos de DynamoDB no es compatible con el uso de operadores numéricos (equivalentes numéricos e intervalo numérico). Incluso si los elementos de la tabla se

almacenan como números, estos parámetros se convierten en cadenas en el objeto de registro JSON.

## Uso del operador Exists

Debido a la forma en que están estructurados los objetos de eventos JSON de DynamoDB, el uso del operador Exists requiere un cuidado especial. El operador Exists solo funciona en los nodos hoja en el evento JSON, por lo que si el patrón de filtro usa Exists para probar la presencia de un nodo intermedio, no funcionará. Considere el siguiente elemento de la tabla de DynamoDB:

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
    {"S": "Sales"},
    {"S": "Marketing"},
    {"S": "Support"}
  ]}
}
```

Es posible que deba crear un patrón de filtro como el siguiente para comprobar si hay eventos que contengan "Organizations":

```
{ "dynamodb" : { "NewItem" : { "Organizations" : [ { "exists": true } ] } } }
```

Sin embargo, este patrón de filtro nunca devolvería una coincidencia porque "Organizations" no es un nodo hoja. El siguiente ejemplo muestra cómo utilizar correctamente el operador Exists para crear el patrón de filtro deseado:

```
{ "dynamodb" : { "NewItem" : { "Organizations": {"L": {"S": [ {"exists":
true } ] } } } } }
```

## Formato JSON para filtrado de DynamoDB

Para filtrar correctamente los eventos de orígenes de DynamoDB, tanto el campo de datos como los criterios de filtro del campo de datos (dynamodb) deben estar en un formato JSON válido. Si

el formato JSON de alguno de los campos no es válido, Lambda elimina el mensaje o genera una excepción. En la siguiente tabla se resume el comportamiento específico:

Formato de los datos entrantes	Formato del patrón de filtro para las propiedades de datos	Acción resultante
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	No JSON	Lambda genera una excepción al crear o actualizar la asignación de origen de eventos. El formato JSON del patrón de filtro de las propiedades de datos debe ser válido.
No JSON	JSON válido	Lambda elimina el registro.
No JSON	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
No JSON	No JSON	Lambda genera una excepción al crear o actualizar la asignación de origen de eventos. El formato JSON del patrón de filtro de las propiedades de datos debe ser válido.



## Tutorial: Uso de AWS Lambda con Amazon DynamoDB Streams

En este tutorial, se crea una función de Lambda para consumir eventos de Amazon DynamoDB Stream.

### Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#), para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

#### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, `zip`) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.

### Creación del rol de ejecución

Cree el [rol de ejecución](#) que concederá a su función permiso para obtener acceso a los recursos de AWS.

## Para crear un rol de ejecución

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Entidad de confianza: Lambda.
  - Permisos: AWSLambdaDynamoDBExecutionRole.
  - Nombre de rol: **lambda-dynamodb-role**.

El AWSLambdaDynamoDBExecutionRole tiene los permisos que la función necesita para leer elementos desde DynamoDB y escribir registros al CloudWatch Logs.

## Creación de la función

Cree una función de Lambda que procese los eventos de DynamoDB. El código de la función escribe algunos de los datos del evento entrante en CloudWatch Logs.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
```

```
[assembly:  
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))  
  
namespace AWSLambda_DDB;  
  
public class Function  
{  
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext  
context)  
    {  
        context.Logger.LogInformation($"Beginning to process  
{dynamoEvent.Records.Count} records...");  
  
        foreach (var record in dynamoEvent.Records)  
        {  
            context.Logger.LogInformation($"Event ID: {record.EventID}");  
            context.Logger.LogInformation($"Event Name: {record.EventName}");  
  
            context.Logger.LogInformation(JsonSerializer.Serialize(record));  
        }  
  
        context.Logger.LogInformation("Stream processing complete.");  
    }  
}
```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
package main
```

```
import (
    "context"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-lambda-go/events"
    "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
error) {
    if len(event.Records) == 0 {
        return nil, fmt.Errorf("received empty event")
    }

    for _, record := range event.Records {
        LogDynamoDBRecord(record)
    }

    message := fmt.Sprintf("Records processed: %d", len(event.Records))
    return &message, nil
}

func main() {
    lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
    fmt.Println(record.EventID)
    fmt.Println(record.EventName)
    fmt.Printf("%+v\n", record.Change)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
    com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
    GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
        GSON.toJson(record.getDynamodb()));
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## Consumo de un evento de DynamoDB con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
}

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Consumo de un evento de DynamoDB con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:". $eventName. "\n");
            $this->logger->info("Keys:". json_encode($keys). "\n");
            $this->logger->info("Old Image:". json_encode($old). "\n");
            $this->logger->info("New Image:". json_encode($new));
        }
    }
}
```

```
        // TODO: Do interesting work based on the new data

        // Any exception thrown will be logged and the invocation will be
marked as failed
    }

    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords items");
}
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```



## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event:, context:)
  return 'received empty event' if event['Records'].empty?

  event['Records'].each do |record|
    log_dynamodb_record(record)
  end

  "Records processed: #{event['Records'].length}"
end

def log_dynamodb_record(record)
  puts record['eventID']
  puts record['eventName']
  puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
    ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }
}
```



```
--role arn:aws:iam::111122223333:role/lambda-dynamodb-role
```

## Probar la función de Lambda

En este paso, invocará la función de Lambda manualmente mediante el comando `invoke` de la CLI de AWS Lambda y el siguiente evento de muestra de DynamoDB. Copie lo siguiente en un archivo denominado `input.txt`.

### Example input.txt

```
{
  "Records": [
    {
      "eventID": "1",
      "eventName": "INSERT",
      "eventVersion": "1.0",
      "eventSource": "aws:dynamodb",
      "awsRegion": "us-east-1",
      "dynamodb": {
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "111",
        "SizeBytes": 26,
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "eventSourceARN": "stream-ARN"
    },
    {
      "eventID": "2",
      "eventName": "MODIFY",
      "eventVersion": "1.0",
```

```
"eventSource":"aws:dynamodb",
"awsRegion":"us-east-1",
"dynamodb":{
  "Keys":{
    "Id":{
      "N":"101"
    }
  },
  "NewImage":{
    "Message":{
      "S":"This item has changed"
    },
    "Id":{
      "N":"101"
    }
  },
  "OldImage":{
    "Message":{
      "S":"New item!"
    },
    "Id":{
      "N":"101"
    }
  },
  "SequenceNumber":"222",
  "SizeBytes":59,
  "StreamViewType":"NEW_AND_OLD_IMAGES"
},
"eventSourceARN":"stream-ARN"
},
{
  "eventID":"3",
  "eventName":"REMOVE",
  "eventVersion":"1.0",
  "eventSource":"aws:dynamodb",
  "awsRegion":"us-east-1",
  "dynamodb":{
    "Keys":{
      "Id":{
        "N":"101"
      }
    }
  },
  "OldImage":{
    "Message":{
```

```
        "S":"This item has changed"
      },
      "Id":{
        "N":"101"
      }
    },
    "SequenceNumber":"333",
    "SizeBytes":38,
    "StreamViewType":"NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN":"stream-ARN"
}
]
```

Ejecute el siguiente comando de la `invoke`.

```
aws lambda invoke --function-name ProcessDynamoDBRecords \  
  --cli-binary-format raw-in-base64-out \  
  --payload file://input.txt outputfile.txt
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

La función devuelve la cadena `message` en el cuerpo de la respuesta.

Verifique la salida en el archivo `outputfile.txt`.

## Crear una tabla de DynamoDB con un flujo habilitado

Crear una tabla de Amazon DynamoDB con un flujo habilitado.

Para crear una tabla de DynamoDB

1. Abra la [consola de DynamoDB](#).
2. Elija Crear tabla.
3. Cree una tabla con la siguiente configuración:
  - Table name (Nombre de la tabla) – **lambda-dynamodb-stream**

- Clave principal: **id** (cadena)
4. Seleccione Crear.

### Habilitar secuencias

1. Abra la [consola de DynamoDB](#).
2. Elija Tables (Tablas).
3. Elija la tabla lambda-dynamodb-stream (Flujo de dynamodb lambda).
4. En Exports and streams (Exportaciones y flujos), elija DynamoDB stream details (Detalles del flujo de DynamoDB).
5. Elija Turn on.
6. En Tipo de vista, elija Solo atributos clave.
7. Seleccione Activar flujo.

Anote el ARN del flujo. Lo necesitará en el siguiente paso al asociar el flujo a la función de Lambda. Para obtener más información acerca de cómo habilitar flujos, consulte [Captura de la actividad de las tablas con DynamoDB Streams](#).

## Añadir un origen de eventos en AWS Lambda

Crear un mapeo de origen de eventos en AWS Lambda. Esta asignación de orígenes de eventos asocia el flujo de DynamoDB a la función de Lambda. Una vez creado este mapeo de origen de eventos, AWS Lambda comienza a sondear el flujo.

Ejecute el siguiente comando `create-event-source-mapping` de la AWS CLI. Después de ejecutar el comando, anote el UUID. Necesitará este UUID para hacer referencia al mapeo de origen de eventos en los comandos, por ejemplo, al eliminar el mapeo de origen de eventos.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \  
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Esto crea un mapeo entre el flujo de DynamoDB especificado y la función de Lambda. Puede asociar un flujo de DynamoDB a varias funciones de Lambda, así como asociar la misma función de Lambda a varios flujos. Sin embargo, las funciones de Lambda compartirán el rendimiento de lectura para el flujo que comparten.

Para obtener la lista de mapeos de orígenes de eventos, ejecute el siguiente comando.

```
aws lambda list-event-source-mappings
```

La lista devuelve todos los mapeos de orígenes de eventos creados, y para cada uno de ellos muestra el `LastProcessingResult`, entre otras cosas. Este campo se utiliza para proporcionar un mensaje informativo en caso de que surja algún problema. Los valores como `No records processed` (indica que AWS Lambda no ha comenzado el sondeo o que no hay registros en el flujo) y `OK` (indica que AWS Lambda ha leído correctamente los registros del flujo y ha invocado la función de Lambda) indican que no hay ningún problema. Si hay algún problema, recibirá un mensaje de error.

Si tiene muchos mapeos de origen de eventos, use la función `nombrar` parámetro para reducir los resultados.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

## Prueba de la configuración

Probar la experiencia integral. A medida que se actualiza la tabla, DynamoDB escribe los registros de eventos en el flujo. Cuando AWS Lambda sondea el flujo, detecta nuevos registros en él e invoca la función de Lambda en nombre del usuario pasando eventos a esta.

1. En la consola de DynamoDB, agregue, actualice y elimine elementos de la tabla. DynamoDB escribe registros de estas acciones en el flujo.
2. AWS Lambda sondea el flujo y, cuando detecta actualizaciones en este, invoca la función de Lambda pasando los datos de eventos que encuentra en el flujo.
3. La función se ejecuta y crea registros en Amazon CloudWatch. Puede verificar los registros reportados en la consola de Amazon CloudWatch.

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.



2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete(Eliminar).

#### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

#### Para eliminar la tabla de DynamoDB

1. Abra la página [Tables \(Tablas\)](#) en la consola de DynamoDB.
2. Seleccione la tabla que ha creado.
3. Elija Eliminar.
4. Escriba **delete** en el cuadro de texto.
5. Elija Delete table (Eliminar tabla).

# Procese los eventos del ciclo de vida de Amazon EC2 con una función de Lambda

Puede utilizar AWS Lambda para procesar eventos del ciclo de vida desde Amazon Elastic Compute Cloud y administrar los recursos de Amazon EC2. Amazon EC2 envía eventos a [Amazon EventBridge \(Eventos de CloudWatch\)](#) para [eventos del ciclo de vida](#), como cuando una instancia cambia de estado, cuando se completa una instantánea de volumen de Amazon Elastic Block Store o cuando se programa que se termine una instancia de spot. Configura EventBridge (CloudWatch Events) para reenviar esos eventos a una función de Lambda para su procesamiento.

EventBridge (CloudWatch Events) invoca su función de Lambda de forma asíncrona con el documento de evento de Amazon EC2.

## Example Ciclo de vida de la instancia

```
{
  "version": "0",
  "id": "b6ba298a-7732-2226-xmpl-976312c1a050",
  "detail-type": "EC2 Instance State-change Notification",
  "source": "aws.ec2",
  "account": "111122223333",
  "time": "2019-10-02T17:59:30Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ec2:us-east-1:111122223333:instance/i-0c314xmplcd5b8173"
  ],
  "detail": {
    "instance-id": "i-0c314xmplcd5b8173",
    "state": "running"
  }
}
```

Para obtener detalles sobre la configuración de eventos, consulte [Invocación de una función de Lambda según una programación](#). Para ver una función de ejemplo que procesa las notificaciones de instantáneas de Amazon EBS, consulte [EventBridge Scheduler para Amazon EBS](#).

También puede utilizar AWS SDK para administrar instancias y otros recursos con la API de Amazon EC2.

## Cómo otorgar permisos a EventBridge (Eventos de CloudWatch)

Para procesar eventos del ciclo de vida desde Amazon EC2, EventBridge (CloudWatch Events) necesita permiso para invocar su función. Este permiso proviene de la [política basada en recursos](#) de la función. Si utiliza la consola de EventBridge (CloudWatch Events) para configurar un desencadenador de eventos, la consola actualiza la política basada en recursos en su nombre. De lo contrario, agregue una declaración como la siguiente:

Example Instrucción de una política basada en recursos para notificaciones del ciclo de vida de Amazon EC2

```
{
  "Sid": "ec2-events",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:us-east-1:12456789012:function:my-function",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:events:us-east-1:12456789012:rule/*"
    }
  }
}
```

Para agregar una instrucción, utilice el comando de la AWS CLI `add-permission`.

```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
--principal events.amazonaws.com --function-name my-function --source-arn
'arn:aws:events:us-east-1:12456789012:rule/*'
```

Si la función utiliza el AWS SDK para administrar recursos de Amazon EC2, agregue permisos de Amazon EC2 al [rol de ejecución](#) de la función.

# Procese las solicitudes de Application Load Balancer con Lambda

Puede utilizar una función de Lambda para procesar solicitudes de un balanceador de carga de aplicaciones. Elastic Load Balancing admite funciones de Lambda como destino para un balanceador de carga de aplicaciones. Utilice reglas de balanceador de carga para direccionar solicitudes HTTP a una función según la ruta o los valores de encabezado. Procese la solicitud y devuelva una respuesta HTTP desde de su función de Lambda.

Elastic Load Balancing invoca la función de Lambda de forma sincrónica con un evento que contiene el cuerpo de la solicitud y metadatos.

Example Balanceador de carga de aplicaciones solicita un evento

```
{
  "requestContext": {
    "elb": {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
east-1:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
    }
  },
  "httpMethod": "GET",
  "path": "/lambda",
  "queryStringParameters": {
    "query": "1234ABCD"
  },
  "headers": {
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8",
    "accept-encoding": "gzip",
    "accept-language": "en-US,en;q=0.9",
    "connection": "keep-alive",
    "host": "lambda-alb-123578498.us-east-1.elb.amazonaws.com",
    "upgrade-insecure-requests": "1",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http",
    "x-imforwards": "20"
  },
  "body": ""
}
```

```
"isBase64Encoded": False
}
```

Su función procesa el evento y devuelve un documento de respuesta al balanceador de carga en JSON. Elastic Load Balancing convierte el documento en una respuesta HTTP de éxito o error y lo devuelve al usuario.

#### Example formato del documento de respuesta

```
{
  "statusCode": 200,
  "statusDescription": "200 OK",
  "isBase64Encoded": False,
  "headers": {
    "Content-Type": "text/html"
  },
  "body": "<h1>Hello from Lambda!</h1>"
}
```

Para configurar un balanceador de carga de aplicaciones como desencadenador de funciones, conceda a Elastic Load Balancing permiso para ejecutar la función, cree un grupo de destino que envíe las solicitudes a la función y agregue una regla al balanceador de carga para enviar las solicitudes al grupo de destino.

Utilice el comando `add-permission` para añadir una instrucción de permiso a la política basada en recursos de la función.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

Debería ver los siguientes datos de salida:

```
{
  "Statement": "{\"Sid\":\"load-balancer\",\"Effect\":\"Allow\",\"Principal\":\n{\"Service\":\"elasticloadbalancing.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\n\",\"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:alb-function\"}"
}
```

Para ver instrucciones sobre cómo configurar el agente de escucha de balanceador de carga de aplicaciones, consulte [Funciones de Lambda como destino](#) en la Guía del usuario para balanceadores de carga de aplicaciones.

# Invocación de una función de Lambda según una programación

El [Programador de Amazon EventBridge](#) es un programador sin servidor que le permite crear, ejecutar y administrar tareas desde un servicio administrado y centralizado. Con el Programador de EventBridge, puede crear programadores mediante expresiones cron y rate para patrones recurrentes, o configurar invocaciones únicas. Puede configurar intervalos de tiempo flexibles para la entrega, definir límites de reintentos y establecer el tiempo máximo de retención para los eventos sin procesar.

Al configurar el Programador de EventBridge con Lambda, el Programador de EventBridge invoca la función de Lambda de forma asíncrona. En esta página, se explica cómo utilizar el Programador de EventBridge para invocar una función de Lambda según una programación.

## Configurar el rol de ejecución

Al crear una programación nueva, el Programador de EventBridge debe tener permiso para invocar la operación de la API de destino en su nombre. Estos permisos se conceden al Programador de EventBridge mediante un rol de ejecución. La política de permisos que adjunta a la función de ejecución de su programación define los permisos necesarios. Estos permisos dependen de la API de destino que quiera que invoque el Programador de EventBridge.

Al utilizar la consola del Programador de EventBridge para crear una programación, como en el siguiente procedimiento, el Programador de EventBridge configura de forma automática un rol de ejecución en función del destino seleccionado. Si desea crear una programación con uno de los SDK del Programador de EventBridge, la AWS CLI o AWS CloudFormation, debe tener un rol de ejecución existente que conceda los permisos que el Programador de EventBridge requiere para invocar un destino. A fin de obtener más información sobre cómo configurar de forma manual un rol de ejecución para su programación, consulte [Setting up the execution role](#) en EventBridge Scheduler User Guide.

## Crear una programación

Para crear una programación con la consola, realice lo siguiente:

1. Abra la consola del Programador de Amazon EventBridge en <https://console.aws.amazon.com/scheduler/home>.
2. En la página de Programaciones, elija Crear programación.

3. En la página de Especificar los detalles de la programación, en la sección de Nombre y descripción de la programación, realice lo siguiente:
  - a. En Nombre de la programación, escriba un nombre para la programación. Por ejemplo, **MyTestSchedule**.
  - b. (Opcional) En Descripción, escriba una descripción para su programación. Por ejemplo, **My first schedule**.
  - c. En Grupo de programaciones, elija un grupo de programaciones de la lista desplegable. Si no tiene un grupo, elija predeterminado. Para crear un grupo de programaciones, elija crear mi propia programación.

Los grupos de programaciones se utilizan para agregar etiquetas a grupos de programaciones.

4. • Elija sus opciones de programación.

Ocurrencia	Haga lo siguiente...
<p>Programación única</p> <p>Una programación única invoca solo una vez un objetivo en la fecha y hora que especifique.</p>	<p>En Fecha y hora, realice lo siguiente:</p> <ul style="list-style-type: none"> <li>• Ingrese una fecha válida en el formato YYYY/MM/DD .</li> <li>• Ingrese una marca de tiempo en el formato hh :mm de 24 horas.</li> <li>• En Zona horaria, elija la zona horaria.</li> </ul>
<p>Programación recurrente</p> <p>Una programación recurrente invoca un objetivo a una velocidad que especifique mediante una expresión cron o rate.</p>	<p>a. En Tipo de programación, realice una de las siguientes acciones:</p> <ul style="list-style-type: none"> <li>• Para utilizar una expresión Cron para definir la programac</li> </ul>



Ocurrencia	Haga lo siguiente...	
	<p>ión, elija Programación basada en Cron e ingrese la expresión Cron.</p> <ul style="list-style-type: none"><li>• Para utilizar una expresión de frecuencia para definir la programación, elija Programación basada en la frecuencia e ingrese la expresión de frecuencia.</li></ul> <p>Para obtener más información sobre las expresiones cron y rate, consulte <a href="#">Schedule types on EventBridge Scheduler</a> en Amazon EventBridge Scheduler User Guide.</p> <p>b. En Intervalo de tiempo flexible, elija Apagado para desactivar la opción o elegir uno de los periodos de tiempo predefinidos. Por ejemplo, si elige 15 minutos y establece una programación recurrente para invocar su objetivo una vez cada hora,</p>	

Ocurrencia	Haga lo siguiente...
	el horario se ejecuta 15 minutos después del inicio de cada hora.

5. (Opcional) Si elige Programación recurrente en el paso anterior, en la sección de Periodo de tiempo, realice lo siguiente:
  - a. En Zona horaria, elija una zona horaria.
  - b. En Fecha y hora de inicio, ingrese una fecha válida en el formato YYYY/MM/DD y, a continuación, especifique una marca de tiempo en el formato hh:mm de 24 horas.
  - c. En Fecha y hora de finalización, ingrese una fecha válida en el formato YYYY/MM/DD y, a continuación, especifique una marca de tiempo en el formato hh:mm de 24 horas.
6. Seleccione Siguiente.
7. En la página Seleccionar destino, elija la operación de la API de AWS que invoca el Programador de EventBridge:
  - a. Elija Invocación de AWS Lambda.
  - b. En la sección de Invocar, seleccione una función o elija Crear función de Lambda nueva.
  - c. (Opcional) Ingrese una carga útil de JSON. Si no ingresa una carga útil, el Programador de EventBridge utilizará un evento vacío para invocar la función.
8. Elija Siguiente.
9. En la página Configuración, haga lo siguiente:
  - a. Para activar la programación, en Estado de la programación, cambie a Habilitar programación.
  - b. A fin de configurar una política de reintentos para su programación, en Política de reintento y cola de mensajes fallidos (DLQ), realice lo siguiente:
    - Cambie a Reintentar.
    - En Antigüedad máxima del evento, ingrese el máximo de horas y minutos que el Programador de EventBridge debe mantener un evento sin procesar.
    - El tiempo máximo es de 24 horas.
    - En Cantidad máxima de reintentos, ingrese el número máximo de veces que el Programador de EventBridge reintentará la programación si el objetivo devuelve un error.

El valor máximo es 185 reintentos.

Con las políticas de reintentos, si un programa no puede invocar su objetivo, el Programador de EventBridge vuelve a ejecutar el programa. Si se encuentra configurado, debe establecer el tiempo máximo de retención y los reintentos máximos para la programación.

- c. Elija dónde almacena los eventos no entregados el Programador de EventBridge.

Opción Cola de mensajes fallidos (DLQ)	Haga lo siguiente...
No almacenar	Seleccione Ninguno.
Guardar el evento en la misma Cuenta de AWS donde crea la programación	<ol style="list-style-type: none"> <li>Elija Seleccionar una cola de Amazon SQS en mi Cuenta de AWS como DLQ.</li> <li>Elija el Nombre de recurso de Amazon (ARN) para la cola de Amazon SQS.</li> </ol>
Guardar el evento en una Cuenta de AWS diferente de donde crea la programación	<ol style="list-style-type: none"> <li>Elija Especificar una cola de Amazon SQS en otras Cuentas de AWS como DLQ.</li> <li>Ingrese el Nombre de recurso de Amazon (ARN) para la cola de Amazon SQS.</li> </ol>

- d. Para utilizar una clave administrada por el cliente a fin de cifrar la entrada de destino, en Cifrado, elija Personalizar la configuración de cifrado (avanzado).

Si elige esta opción, ingrese un ARN de clave de KMS existente o elija Crear una AWS KMS key para navegar hasta la consola de AWS KMS. Para obtener más información sobre

cómo el Programador de EventBridge cifra los datos en reposo, consulte [Encryption at rest](#) en Amazon EventBridge Scheduler User Guide.

- e. Para que el Programador de EventBridge cree un rol de ejecución nuevo en su nombre, elija Crear un nuevo rol para esta programación. A continuación, ingrese un nombre para el Nombre de rol. Si elige esta opción, el Programador de EventBridge adjunta al rol los permisos necesarios para el objetivo creado con la plantilla.

10. Elija Siguiente.

11. En la página de Revisar y crear una programación, revise los detalles de su programación. En cada sección, elija Editar para volver a ese paso y editar sus detalles.

12. Elija Crear programación.

Puede ver una lista de sus programaciones nuevas y existentes en la página de Programaciones. En la columna de Estado, verifique que su programación nueva se encuentre Habilitada.

Para confirmar que el Programador de EventBridge ha invocado la función, [compruebe Registros de Amazon CloudWatch de la función](#).

## Recursos relacionados

Para obtener más información sobre el Programador de EventBridge, consulte lo siguiente:

- [EventBridge Scheduler User Guide](#)
- [Referencia de la API del Programador de EventBridge](#)
- [Precios del Programador de EventBridge](#)

## Uso de AWS Lambda con AWS IoT

AWS IoT proporciona una comunicación segura entre dispositivos conectados a Internet (como sensores) y la nube de AWS. Esto le permite recopilar, almacenar y analizar los datos de telemetría de varios dispositivos.

Puede crear reglas de AWS IoT para que sus dispositivos interactúen con los servicios de AWS. El [motor de reglas](#) de AWS IoT proporciona un lenguaje basado en SQL para seleccionar datos de cargas de mensajes y enviar los datos a otros servicios como Amazon S3, Amazon DynamoDB y AWS Lambda. Cuando quiera invocar otro servicio de AWS o un servicio de terceros, defina una regla para invocar una función de Lambda.

Cuando un mensaje de IoT entrante desencadena la regla, AWS IoT invoca su función de Lambda [de forma asíncrona](#) y pasa los datos del mensaje de IoT a la función.

El siguiente ejemplo muestra una lectura de humedad del sensor de un invernadero. Los valores row y pos identifican la ubicación del sensor. Este evento de ejemplo se basa en el tipo de invernadero de los [Tutoriales de reglas de AWS IoT](#).

Example Evento de mensaje de AWS IoT

```
{
  "row" : "10",
  "pos" : "23",
  "moisture" : "75"
}
```

En la invocación asíncrona, Lambda pone en cola el mensaje y, si la función devuelve un error, [lo intenta de nuevo](#). Configure su función con un [destination](#) para retener eventos que su función no pudo procesar.

Debe conceder permiso para que el servicio AWS IoT invoque su función de Lambda. Utilice el comando `add-permission` para añadir una instrucción de permiso a la política basada en recursos de la función.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
iot.amazonaws.com
```

Debería ver los siguientes datos de salida:

```
{
  "Statement": "{\\"Sid\\":\\"iot-events\\",\\"Effect\\":\\"Allow\\",\\"Principal\\":
  {\\"Service\\":\\"iot.amazonaws.com\\"},\\"Action\\":\\"lambda:InvokeFunction\\",\\"Resource\\":
  \\"arn:aws:lambda:us-east-1:123456789012:function:my-function\\"}"
}
```

Para obtener más información acerca de cómo utilizar Lambda con AWS IoT, consulte [Creación de una regla de AWS Lambda](#).

# Cómo Lambda procesa los registros de Amazon Kinesis Data Streams

Puede utilizar una función de Lambda para procesar los registros de un [flujo de datos de Amazon Kinesis](#). Puede asignar una función de Lambda a un consumidor de rendimiento compartido (iterador estándar) de Kinesis Data Streams o a un consumidor de rendimiento dedicado con [distribución ramificada mejorada](#). Para iteradores estándar, Lambda sondea cada partición de la secuencia de Kinesis en busca de registros utilizando el protocolo HTTP. El mapeo de origen de eventos comparte el rendimiento de lectura con otros consumidores de la partición.

Para obtener información detallada sobre los flujos de datos de Kinesis, consulte [Lectura de datos de Amazon Kinesis Data Streams](#).

## Note

Kinesis cobra por cada partición y, para una distribución ramificada mejorada, por los datos leídos desde el flujo. Para obtener más información sobre precios, consulte [Precios de Amazon Kinesis](#).

## Temas

- [Flujos de sondeo y procesamiento por lotes](#)
- [Evento de ejemplo](#)
- [Procesamiento de registros de Amazon Kinesis Data Streams](#)
- [Configuración de la respuesta por lotes parcial con Kinesis Data Streams y Lambda](#)
- [Conserve los registros de lotes descartados para un origen de eventos de Kinesis Data Streams en Lambda](#)
- [Implementación del procesamiento con estado de Kinesis Data Streams en Lambda](#)
- [Parámetros de Lambda para las asignaciones de orígenes de eventos de Amazon Kinesis Data Streams](#)
- [Uso del filtrado de eventos con una fuente de eventos de Kinesis](#)
- [Tutorial: Uso de Lambda con Kinesis Data Streams](#)

## Flujos de sondeo y procesamiento por lotes

Lambda lee los registros del flujo de datos e invoca la función [sincrónicamente](#) con un evento que contiene registros de flujo. Lambda lee los registros por lotes e invoca la función para procesar los registros del lote. Cada lote contiene registros de una única partición o flujo de datos.

Para los flujos de datos estándar de Kinesis, Lambda sondea cada partición en el flujo para buscar registros una vez por segundo. En el caso de la [distribución mejorada de Kinesis](#), Lambda utiliza una conexión HTTP/2 para escuchar los registros que se envían desde Kinesis. Cuando hay registros disponibles, Lambda invoca la función y espera el resultado.

De forma predeterminada, Lambda invoca su función tan pronto como los registros estén disponibles. Si el lote que Lambda lee del origen de eventos solo tiene un registro, Lambda envía solo un registro a la función. Para evitar invocar la función con un número de registros pequeño, puede indicar al origen de eventos que almacene en búfer registros durante hasta 5 minutos configurando un plazo de procesamiento por lotes. Antes de invocar la función, Lambda continúa leyendo los registros del origen de eventos hasta que haya recopilado un lote completo, venza el plazo de procesamiento por lotes o el lote alcance el límite de carga de 6 MB. Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

### Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

Lambda no espera a que se completen las [extensiones](#) configuradas antes de enviar el siguiente lote para su procesamiento. En otras palabras, las extensiones pueden seguir ejecutándose mientras Lambda procesa el siguiente lote de registros. Esto puede provocar problemas de limitación si infringe alguno de los ajustes o límites de [simultaneidad](#) de la cuenta. Para detectar si se trata de un posible problema, supervise sus funciones y compruebe si ve [métricas de simultaneidad](#) más elevadas de lo esperado para la asignación de orígenes de eventos. Debido a los tiempos cortos entre invocaciones, Lambda puede informar brevemente un uso de simultaneidad superior al número de particiones. Esto puede ser cierto incluso para las funciones de Lambda sin extensiones.



Defina la configuración [ParallelizationFactor](#) para procesar una partición de un flujo de datos de Kinesis con más de una invocación de Lambda simultáneamente. Puede especificar el número de lotes simultáneos que Lambda sondea desde una partición a través de un factor de paralelización de 1 (predeterminado) a 10. Por ejemplo, cuando establece `ParallelizationFactor` en 2, puede tener un máximo de 200 invocaciones de Lambda simultáneas para procesar 100 particiones de datos de Kinesis (aunque, en la práctica, es posible que observe diferentes valores para la métrica `ConcurrentExecutions`). Esto ayuda a escalar verticalmente el rendimiento de procesamiento cuando el volumen de datos es volátil y el `IteratorAge` es alto. Cuando aumenta el número de lotes simultáneos por partición, Lambda sigue garantizando el procesamiento en orden a nivel de clave de partición.

También puede utilizar `ParallelizationFactor` con la agregación de Kinesis. El comportamiento de la asignación de orígenes de eventos depende de si utiliza la [distribución ramificada mejorada](#):

- Sin distribución ramificada mejorada: todos los eventos incluidos en un evento agregado deben tener la misma clave de partición. La clave de partición también debe coincidir con la del evento agregado. Si los eventos incluidos en el evento agregado tienen claves de partición diferentes, Lambda no puede garantizar el procesamiento de los eventos ordenados por clave de partición.
- Con distribución ramificada mejorada: en primer lugar, Lambda decodifica el evento agregado en sus eventos individuales. El evento agregado puede tener una clave de partición diferente a la de los eventos que contiene. Sin embargo, los eventos que no se corresponden con la clave de partición [se eliminan y se pierden](#). Lambda no procesa estos eventos ni los envía a un destino de error configurado.

## Evento de ejemplo

### Example

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
      },
    },
  ],
}
```

```

        "eventSource": "aws:kinesis",
        "eventVersion": "1.0",
        "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
        "eventName": "aws:kinesis:record",
        "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
        "awsRegion": "us-east-2",
        "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
    },
    {
        "kinesis": {
            "kinesisSchemaVersion": "1.0",
            "partitionKey": "1",
            "sequenceNumber":
"49590338271490256608559692540925702759324208523137515618",
            "data": "VGhpcyBpcyBvbmx5IGEdGVzdC4=",
            "approximateArrivalTimestamp": 1545084711.166
        },
        "eventSource": "aws:kinesis",
        "eventVersion": "1.0",
        "eventID":
"shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
        "eventName": "aws:kinesis:record",
        "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
        "awsRegion": "us-east-2",
        "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
    }
]
}

```

## Procesamiento de registros de Amazon Kinesis Data Streams

Para procesar los registros de Amazon Kinesis Data Streams con Lambda, cree un consumidor para el flujo y, a continuación, cree una asignación de orígenes de eventos de Lambda.

### Configurar su flujo de datos y función.

Su función de Lambda es una aplicación consumidora para su flujo de datos. Procesa un lote de registros a la vez desde cada partición. Puede asignar una función Lambda a un consumidor de rendimiento compartido (iterador estándar) o a un consumidor de rendimiento dedicado con distribución ramificada mejorada.

- Iterador estándar: Lambda sondea cada partición del flujo de Kinesis y busca registros a una velocidad base de una vez por segundo. Cuando hay más registros disponibles, Lambda sigue procesando lotes hasta que la función se pone al día con el flujo. El mapeo de origen de eventos comparte el rendimiento de lectura con otros consumidores de la partición.
- Para minimizar la latencia y maximizar el rendimiento de lectura, cree un consumidor de flujo de datos con distribución ramificada mejorada. Los consumidores con distribución ramificada mejorada obtienen una conexión dedicada a cada partición que no afecta a las demás aplicaciones que leen el flujo. Los consumidores de flujos utilizan HTTP/2 para reducir la latencia enviando los registros a Lambda a través de una conexión de larga duración y mediante la compresión de los encabezados de las solicitudes. Es posible crear un consumidor de flujos con la API [RegisterStreamConsumer](#) de Kinesis.

```
aws kinesis register-stream-consumer \  
--consumer-name con1 \  
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Debería ver los siguientes datos de salida:

```
{  
  "Consumer": {  
    "ConsumerName": "con1",  
    "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/  
consumer/con1:1540591608",  
    "ConsumerStatus": "CREATING",  
    "ConsumerCreationTimestamp": 1540591608.0  
  }  
}
```

Para incrementar la velocidad con la que la función procesa los registros, [agregue particiones al flujo de datos](#). Lambda procesa registros en cada partición en orden. Deja de procesar registros adicionales en una partición si la función devuelve un error. Al haber más particiones, se procesan más lotes simultáneamente, lo que reduce el impacto de los errores de simultaneidad.

Si la función no puede aumentar para administrar el número total de lotes simultáneos, [solicite un aumento de cuota](#) o [reserve la simultaneidad](#) para la función.

## Creación de una asignación de orígenes de eventos para invocar la función de Lambda

Para invocar la función de Lambda con registros del flujo de datos, cree una [asignación de orígenes de eventos](#). Puede crear varias asignaciones de orígenes de eventos para procesar los mismos datos con distintas funciones de Lambda o para procesar elementos de varios flujos de datos con una sola función. Al procesar elementos de múltiples flujos de datos, cada lote solo contendrá registros de una única partición o flujo.

Puede configurar las asignaciones de orígenes de eventos para procesar los registros de un flujo en una Cuenta de AWS diferente. Para obtener más información, consulte [the section called “Asignaciones entre cuentas”](#).

Antes de crear una asignación de orígenes de eventos, debe dar permiso a la función de Lambda para leer desde un flujo de datos de Kinesis. Lambda necesita los siguientes permisos para administrar los recursos relacionados con el flujo de datos de Kinesis:

- [kinesis:DescribeStream](#)
- [kinesis:DescribeStreamSummary](#)
- [kinesis:GetRecords](#)
- [kinesis:GetShardIterator](#)
- [kinesis:ListShards](#)
- [kinesis:ListStreams](#)
- [kinesis:SubscribeToShard](#)

La política administrada de AWS [AWSLambdaKinesisExecutionRole](#) incluye estos permisos. Agregue esta política administrada a la función, tal como se describe en el siguiente procedimiento.

### AWS Management Console

#### Cómo añadir permisos de Kinesis a la función

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. En la pestaña Configuración, elija Permisos.
3. En el panel de Roles de ejecución, en Nombre del rol, elija el enlace al rol de ejecución de la función. Este enlace abre la página para ese rol en la consola de IAM.

4. En el panel Políticas de permisos, elija Agregar permisos y, a continuación, elija Adjuntar políticas.
5. En el campo de búsqueda, escriba **AWSLambdaKinesisExecutionRole**.
6. Seleccione la casilla situada junto a la política y elija Añadir permisos.

## AWS CLI

### Cómo añadir permisos de Kinesis a la función

- Ejecute el siguiente comando de la CLI para adjuntar la política de **AWSLambdaKinesisExecutionRole** al rol de ejecución de la función:

```
aws iam attach-role-policy \  
--role-name MyFunctionRole \  
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

## AWS SAM

### Cómo añadir permisos de Kinesis a la función

- En la definición de la función, agregue la propiedad **Policies**, tal como se muestra en el siguiente ejemplo:

```
Resources:  
  MyFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: ./my-function/  
      Handler: index.handler  
      Runtime: nodejs20.x  
      Policies:  
        - AWSLambdaKinesisExecutionRole
```

Una vez que haya configurado los permisos necesarios, cree la asignación de orígenes de eventos.

## AWS Management Console

### Cómo crear la asignación de orígenes de eventos para Kinesis

1. Abra la [página Funciones](#) de la consola de Lambda y seleccione su función.
2. En el panel Información general de la función, elija Agregar desencadenador.
3. En Configuración del desencadenador, para el origen, seleccione Kinesis.
4. Seleccione el flujo de Kinesis para el que quiere crear la asignación de orígenes de eventos y, si lo desea, un consumidor del flujo.
5. (Opcional) Edite el tamaño del lote, la posición inicial y la ventana del lote para la asignación de orígenes de eventos.
6. Elija Añadir.

Al crear la asignación de orígenes de eventos desde la consola, el rol de IAM debe tener los permisos [kinesis:ListStreams](#) y [kinesis:ListStreamConsumers](#).

## AWS CLI

### Cómo crear la asignación de orígenes de eventos de Kinesis

- Ejecute el siguiente comando de la CLI para crear una asignación de orígenes de eventos de Kinesis. Elija su propio tamaño de lote y posición inicial según su caso de uso.

```
aws lambda create-event-source-mapping \  
--function-name MyFunction \  
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \  
--starting-position LATEST \  
--batch-size 100
```

Para especificar una ventana de procesamiento por lotes, agregue la opción `--maximum-batching-window-in-seconds`. Para obtener más información sobre el uso de este u otros parámetros, consulte [create-event-source-mapping](#) en la Referencia de comandos de la AWS CLI.

## AWS SAM

### Cómo crear la asignación de orígenes de eventos de Kinesis

- En la definición de la función, agregue la propiedad `KinesisEvent`, tal como se muestra en el siguiente ejemplo:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs20.x
      Policies:
        - AWSLambdaKinesisExecutionRole
      Events:
        KinesisEvent:
          Type: Kinesis
          Properties:
            Stream: !GetAtt MyKinesisStream.Arn
            StartingPosition: LATEST
            BatchSize: 100

  MyKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
```

Para obtener más información sobre cómo crear una asignación de orígenes de eventos para Kinesis Data Streams en AWS SAM, consulte [Kinesis](#) en la Guía para desarrolladores de AWS Serverless Application Model.

### Posición inicial de flujos y sondeo

Tenga en cuenta que el sondeo de flujos durante la creación y las actualizaciones de la asignación de orígenes de eventos es, en última instancia, coherente.

- Durante la creación de la asignación de orígenes de eventos, es posible que se demore varios minutos en iniciar el sondeo de los eventos del flujo.

- Durante las actualizaciones de la asignación de orígenes de eventos, es posible que se demore varios minutos en detener y reiniciar el sondeo de los eventos del flujo.

Este comportamiento significa que, si especifica LATEST como posición inicial del flujo, la asignación de orígenes de eventos podría omitir eventos durante la creación o las actualizaciones. Para garantizar que no se pierda ningún evento, especifique la posición inicial del flujo como TRIM\_HORIZON o AT\_TIMESTAMP.

## Creación de asignaciones de orígenes de eventos entre cuentas

Amazon Kinesis Data Streams admite [políticas basadas en recursos](#). En consecuencia, puede procesar los datos ingeridos en un flujo en una Cuenta de AWS con una función de Lambda en otra cuenta.

Para crear una asignación de orígenes de eventos para la función de Lambda mediante un flujo de Kinesis en otra Cuenta de AWS, debe configurar el flujo mediante una política basada en recursos para conceder a la función de Lambda permiso para leer elementos. Para obtener información sobre cómo configurar la transmisión para permitir el acceso entre cuentas, consulte [Acceso compartido con funciones de AWS Lambda entre cuentas](#) en la Guía para desarrolladores de Amazon Kinesis Streams.

Una vez que haya configurado la transmisión con una política basada en recursos que otorgue a la función de Lambda los permisos necesarios, cree la asignación de orígenes de eventos mediante cualquiera de los métodos descritos en la sección anterior.

Si decide crear la asignación de orígenes de eventos mediante la consola Lambda, pegue el ARN de la transmisión directamente en el campo de entrada. Si quiere especificar un consumidor para la transmisión, al pegar el ARN del consumidor se rellena en forma automática el campo de transmisión.

## Configuración de la respuesta por lotes parcial con Kinesis Data Streams y Lambda

Al consumir y procesar datos de streaming desde un origen de eventos, de forma predeterminada, Lambda comprueba hasta el número de secuencia más alto de un lote solo cuando el lote se ha completado con éxito. Lambda trata todos los demás resultados como un error completo y vuelve a intentar procesar el lote hasta el límite de reintentos. Para permitir éxitos parciales al procesar lotes de una secuencia, active `ReportBatchItemFailures`. Permitir éxitos parciales puede ayudar



a reducir el número de reintentos en un registro, aunque no impide por completo la posibilidad de reintentos en un registro exitoso.

Para activar `ReportBatchItemFailures`, incluya el valor enumerado

**`ReportBatchItemFailures`** en la lista [FunctionResponseTypes](#). Esta lista indica qué tipos de respuesta están habilitados para su función. Puede configurar esta lista al [crear](#) o [actualizar](#) una asignación de orígenes de eventos.

## Sintaxis del informe

Al configurar los informes sobre errores de elementos por lotes, la clase `StreamsEventResponse` se devuelve con una lista de errores de elementos de lote. Puede utilizar un objeto `StreamsEventResponse` para devolver el número de secuencia del primer registro fallido del lote. También puede crear su propia clase personalizada usando la sintaxis de respuesta correcta. La siguiente estructura JSON muestra la sintaxis de respuesta requerida:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": "<SequenceNumber>"
    }
  ]
}
```

### Note

Si la matriz `batchItemFailures` contiene varios elementos, Lambda usa el registro con el número de secuencia más bajo como punto de control. Luego Lambda vuelve a probar todos los registros a partir de ese punto de control.

## Condiciones de éxito y fracaso

Lambda trata un lote como un éxito completo si devuelve cualquiera de los siguientes elementos:

- Una lista `batchItemFailure` vacía
- Una lista `batchItemFailure` nula
- Una `EventResponse` vacía
- Un `EventResponse` nulo

Lambda trata un lote como un error completo si devuelve cualquiera de los siguientes elementos:

- Una cadena `itemIdentifier` vacía
- Una `itemIdentifier` nula
- Un `itemIdentifier` con un mal nombre de clave

Lambda reintenta fallidos basados en su estrategia de reintento.

## Bisecar un lote

Si su invocación falla y `BisectBatchOnFunctionError` está activada, el lote se divide en bisectos independientemente de su configuración `ReportBatchItemFailures`.

Cuando se recibe una respuesta de éxito parcial de lote y se activan tanto `BisectBatchOnFunctionError` como `ReportBatchItemFailures`, el lote se divide en el número de secuencia devuelto y Lambda vuelve a intentar solo los registros restantes.

Estos son algunos ejemplos de código de función que devuelven la lista de IDs de mensajes fallidos del lote:

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
                item immediately.
                Lambda will immediately begin to retry processing from this
                failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
                    List<StreamsEventResponse.BatchItemFailure>
                    {
```

```

        new StreamsEventResponse.BatchItemFailure
    { ItemIdentifier = record.Kinesis.SequenceNumber }
        }
    };
    }
}
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    return new StreamsEventResponse();
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
}

```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, record := range kinesisEvent.Records {
        curRecordSequenceNumber := ""

        // Process your record
        if /* Your record processing condition here */ {
            curRecordSequenceNumber = record.Kinesis.SequenceNumber
        }

        // Add a condition to check if the record processing failed
        if curRecordSequenceNumber != "" {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": curRecordSequenceNumber})
        }
    }

    kinesisBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return kinesisBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```

        /* Since we are working with streams, we can return the failed
        item immediately.
           Lambda will immediately begin to retry processing from this
        failed item onwards. */
        batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
        return new StreamsEventResponse(batchItemFailures);
    }
}

return new StreamsEventResponse(batchItemFailures);
}
}

```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en [GitHub](#). Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Javascript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.

```

```

        Lambda will immediately begin to retry processing from this failed
        item onwards. */
        return {
            batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
        };
    }
}
console.log(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante TypeScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
    KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<KinesisStreamBatchResponse> => {
    for (const record of event.Records) {

```



```
try {
  logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
  const recordData = await getRecordDataAsync(record.kinesis);
  logger.info(`Record Data: ${recordData}`);
  // TODO: Do interesting work based on the new data
} catch (err) {
  logger.error(`An error occurred ${err}`);
  /* Since we are working with streams, we can return the failed item
  immediately.
     Lambda will immediately begin to retry processing from this failed
  item onwards. */
  return {
    batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
  };
}
logger.info(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}
```

```
// change format for the response
$failures = array_map(
    fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
    $failedRecords
);

return [
    'batchItemFailures' => $failures
];
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
```

```

        return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}

```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Ruby.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
      # immediately.
      # Lambda will immediately begin to retry processing from this failed item
      # onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end
end

```

```

    puts "Successfully processed #{event['Records'].length} records."
    { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end

```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Rust.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
    Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }
}

```

```
for record in &event.payload.records {
    tracing::info!(
        "EventId: {}",
        record.event_id.as_deref().unwrap_or_default()
    );

    let record_processing_result = process_record(record);

    if record_processing_result.is_err() {
        response.batch_item_failures.push(KinesisBatchItemFailure {
            item_identifier: record.kinesis.sequence_number.clone(),
        });
        /* Since we are working with streams, we can return the failed item
immediately.
        Lambda will immediately begin to retry processing from this failed
item onwards. */
        return Ok(response);
    }
}

tracing::info!(
    "Successfully processed {} records",
    event.payload.records.len()
);

Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## Conserve los registros de lotes descartados para un origen de eventos de Kinesis Data Streams en Lambda

La gestión de errores en las asignaciones de orígenes de eventos de Kinesis depende de si el error se produce antes de que se invoque la función o durante la invocación de la función:

- Antes de la invocación: si una asignación de orígenes de eventos de Lambda no puede invocar la función debido a una limitación u otros problemas, lo vuelve a intentar hasta que los registros caduquen o superen la antigüedad máxima configurada en la asignación de orígenes de eventos ([MaximumRecordAgeInSeconds](#)).
- Durante invocación: [si se invoca la función pero devuelve un error, Lambda vuelve a intentarlo hasta que los registros caduquen, superen la antigüedad máxima \(MaximumRecordAgeInSeconds\) o alcancen la cuota de reintento configurada \(MaximumRetryAttempts\)](#). En el caso de errores de función, también puede configurar [BisectBatchOnFunctionError](#), que divide un lote fallido en dos lotes más pequeños, aislando registros fallidos y evitando tiempos de espera. La división de lotes no consume la cuota de reintentos.

Si las medidas de administración de errores fallan, Lambda descarta los registros y continúa procesando lotes del flujo. Con la configuración predeterminada, esto significa que un registro incorrecto puede bloquear el procesamiento en la partición afectada durante un máximo de una semana. Para evitar esto, configure la asignación de orígenes de eventos de su función con un número razonable de reintentos y una antigüedad máxima de registro que se ajuste a su caso de uso.

## Configuración de destinos para invocaciones fallidas

Para retener los registros de las invocaciones de asignación de orígenes de eventos fallidos, agregue un destino a la asignación de orígenes de eventos de su función. Cada registro enviado al destino es un documento JSON con metadatos sobre la invocación fallida. Puede configurar cualquier tema de Amazon SNS o cualquier cola de Amazon SQS como destino. Su rol de ejecución debe tener permisos para el destino:

- Para destinos de SQS: [sqs:SendMessage](#)
- Para destinos SNS: [sns:Publish](#)

Para configurar un destino en caso de error mediante la consola, siga estos pasos:

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En Descripción general de la función, elija Agregar destino.
4. En Origen, elija Invocación de asignación de orígenes de eventos.
5. Para la Asignación de orígenes de eventos, elija un origen de eventos que esté configurado para esta función.
6. En Condición, seleccione En caso de error. Para las invocaciones de asignación de orígenes de eventos, esta es la única condición aceptada.
7. En Tipo de destino, elija el tipo de destino al que Lambda envía los registros de invocación.
8. En Destino, elija un recurso.
9. Seleccione Guardar.

También puede configurar un destino en caso de error mediante la AWS Command Line Interface (AWS CLI). Por ejemplo, el siguiente comando [create-event-source-mapping](#) agrega una asignación de orígenes de eventos con un destino de SQS en caso de error a MyFunction:

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```



El siguiente comando [update-event-source-mapping](#) actualiza una asignación de orígenes de eventos para enviar registros de invocación fallida a un destino de SNS después de dos intentos de reintento, o si los registros tienen más de una hora de antigüedad.

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--maximum-retry-attempts 2 \  
--maximum-record-age-in-seconds 3600 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-  
east-1:123456789012:dest-topic"}}'
```

La configuración actualizada se aplica de forma asincrónica y no se refleja en la salida hasta que se completa el proceso. Utilice el comando [get-event-source-mapping](#) para ver el estado actual.

Para eliminar un destino, introduzca una cadena vacía como argumento del parámetro `destination-config`:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": ""}}'
```

El siguiente ejemplo muestra lo que Lambda envía a un destino de tema de SNS o cola de SQS cuando se produce un error en la invocación de un origen de eventos de Kafka. Como Lambda envía solo los metadatos para estos tipos de destino, utilice los campos `streamArn`, `shardId`, `startSequenceNumber` y `endSequenceNumber` para obtener el registro original completo.

```
{  
  "requestContext": {  
    "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",  
    "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",  
    "condition": "RetryAttemptsExhausted",  
    "approximateInvokeCount": 1  
  },  
  "responseContext": {  
    "statusCode": 200,  
    "executedVersion": "$LATEST",  
    "functionError": "Unhandled"  
  },  
  "version": "1.0",  
  "timestamp": "2019-11-14T00:38:06.021Z",  
  "KinesisBatchInfo": {
```

```
    "shardId": "shardId-000000000001",
    "startSequenceNumber":
"49601189658422359378836298521827638475320189012309704722",
    "endSequenceNumber":
"49601189658422359378836298522902373528957594348623495186",
    "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",
    "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",
    "batchSize": 500,
    "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"
  }
}
```

Puede utilizar esta información para recuperar los registros afectados del flujo para solucionar problemas. Los registros reales no están incluidos, por lo que debe procesar este registro y recuperarlos del flujo antes de que caduquen y se pierdan.

## Implementación del procesamiento con estado de Kinesis Data Streams en Lambda

Las funciones de Lambda pueden ejecutar aplicaciones de procesamiento de flujo continuo. Una secuencia representa datos ilimitados que fluyen de forma continua a través de su aplicación. Para analizar la información de esta entrada de actualización continua, puede enlazar los registros incluidos mediante una ventana definida en términos de tiempo.

Las ventanas de salto constante son ventanas de tiempo distintas que se abren y cierran a intervalos regulares. De forma predeterminada, las invocaciones de Lambda no tienen estado: no se pueden utilizar para procesar datos en múltiples invocaciones continuas sin una base de datos externa. Sin embargo, con las ventanas de salto constante, puede mantener su estado en todas las invocaciones. Este estado contiene el resultado agregado de los mensajes procesados previamente para la ventana actual. Su estado puede ser un máximo de 1 MB por partición. Si supera ese tamaño, Lambda finaliza la ventana antes de tiempo.

Cada registro de una secuencia pertenece a un periodo específico. Lambda procesará cada registro al menos una vez, pero no garantiza que cada registro se procese solo una vez. En casos excepcionales, como el manejo de errores, es posible que algunos registros se procesen más de una vez. Los registros siempre se procesan en orden la primera vez. Si los registros se procesan más de una vez, es posible que lo hagan de forma desordenada.

## Agregación y procesamiento

Su función administrada por el usuario se invoca tanto para la agregación como para procesar los resultados finales de esa agregación. Lambda agrega todos los registros recibidos en la ventana. Puede recibir estos registros en varios lotes, cada uno como una invocación independiente. Cada invocación recibe un estado. Por lo tanto, al usar las ventanas de salto constante, su respuesta de la función de Lambda debe contener una propiedad de `state`. Si la respuesta no contiene una propiedad de `state`, Lambda considera que esto es una invocación fallida. Para satisfacer esta condición, la función puede devolver un objeto de `TimeWindowEventResponse`, que tiene la siguiente forma JSON:

### Example Valores `TimeWindowEventResponse`

```
{
  "state": {
    "1": 282,
    "2": 715
  },
  "batchItemFailures": []
}
```

#### Note

Para las funciones Java, se recomienda utilizar un `Map<String, String>` para representar el estado.

Al final de la ventana, el indicador `isFinalInvokeForWindow` está configurado en `true` para indicar que este es el estado final y que está listo para su procesamiento. Después del procesamiento, la ventana se completa y su invocación final se completa, y luego se elimina el estado.

Al final de la ventana, Lambda utiliza el procesamiento final para las acciones en los resultados de agregación. Su procesamiento final se invoca sincrónicamente. Después de la invocación exitosa, los puntos de control de la función, el número de secuencia y el procesamiento de flujo continúa. Si la invocación no tiene éxito, su función de Lambda suspende el procesamiento posterior hasta una invocación exitosa.

## Example KinesisTimeWindowEvent

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1607497475.000
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",
      "awsRegion": "us-east-1",
      "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-
stream"
    }
  ],
  "window": {
    "start": "2020-12-09T07:04:00Z",
    "end": "2020-12-09T07:06:00Z"
  },
  "state": {
    "1": 282,
    "2": 715
  },
  "shardId": "shardId-000000000006",
  "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream",
  "isFinalInvokeForWindow": false,
  "isWindowTerminatedEarly": false
}
```

## Configuración

Puede configurar ventanas de salto constante al crear o actualizar una asignación de orígenes de eventos. Para configurar una ventana de saltos de tamaño constante, especifique la ventana en

segundos ([TumblingWindowInSeconds](#)). El siguiente comando de ejemplo AWS Command Line Interface (AWS CLI) crea una asignación de origen de eventos de streaming que tiene una ventana de salto constante de 120 segundos. Se nombra la función de Lambda definida para la agregación y el procesamiento se llama `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping \  
--event-source-arn arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream \  
--function-name tumbling-window-example-function \  
--starting-position TRIM_HORIZON \  
--tumbling-window-in-seconds 120
```

Lambda determina los límites de la ventana de salto constante en función de la hora en que se insertaron los registros en la secuencia. Todos los registros tienen una marca de hora aproximada disponible que Lambda utiliza en las determinaciones de límites.

Las agregaciones de ventanas de saltos constantes no admiten el reendurecimiento. Cuando una partición termina, Lambda considera la ventana actual como cerrada y las particiones secundarias comienzan su propia ventana en un estado renovado. Cuando no se agrega ningún registro nuevo a la ventana actual, Lambda espera hasta 2 minutos antes de suponer que la ventana ha terminado. Esto ayuda a garantizar que la función lea todos los registros de la ventana actual, incluso si los registros se agregan de forma intermitente.

Ventanas de saltos constantes son totalmente compatibles con las directivas de reintento existentes `maxRetryAttempts` y `maxRecordAge`.

Example Handler.py: agregación y procesamiento

La siguiente función de Python muestra cómo agregar y luego procesar su estado final:

```
def lambda_handler(event, context):  
    print('Incoming event: ', event)  
    print('Incoming state: ', event['state'])  
  
    #Check if this is the end of the window to either aggregate or process.  
    if event['isFinalInvokeForWindow']:  
        # logic to handle final state of the window  
        print('Destination invoke')  
    else:  
        print('Aggregate invoke')  
  
    #Check for early terminations
```

```

if event['isWindowTerminatedEarly']:
    print('Window terminated early')

#Aggregation logic
state = event['state']
for record in event['Records']:
    state[record['kinesis']['partitionKey']] = state.get(record['kinesis']
['partitionKey'], 0) + 1

print('Returning state: ', state)
return {'state': state}

```

## Parámetros de Lambda para las asignaciones de orígenes de eventos de Amazon Kinesis Data Streams

Todos los tipos de asignación de orígenes de eventos de Lambda comparten las mismas operaciones [CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Amazon Kinesis.

Parámetro	Obligatoria	Predeterminado	Notas
<a href="#">BatchSize</a>	N	100	Máximo: 10 000
<a href="#">BisectBatchOnFunctionError</a>	N	false	Ninguno
<a href="#">DestinationConfig</a>	N	N/A	Destino de tema de la cola de Amazon SQS o de Amazon SNS para registros descartados Para obtener más información, consulte <a href="#">Configuración de destinos para invocaciones fallidas</a> .
<a href="#">Enabled (Habilitado)</a>	N	true	Ninguno

Parámetro	Obligatoria	Predeterminado	Notas
<a href="#">EventSourceArn</a>	Y	N/A	ARN del flujo de datos o un consumidor de flujos
<a href="#">FunctionName</a>	Y	N/A	Ninguno
<a href="#">FunctionResponseTypes</a>	N	N/A	Para permitir que la función informe de errores específicos de un lote, incluya el valor <code>ReportBatchItemFailures</code> en <code>FunctionResponseTypes</code> . Para obtener más información, consulte <a href="#">Configuración de la respuesta por lotes parcial con Kinesis Data Streams y Lambda</a> .
<a href="#">MaximumBatchingWindowInSeconds</a>	N	0	Ninguno
<a href="#">MaximumRecordAgeInSeconds</a>	N	-1	-1 significa infinito: Lambda no descarta registros (se sigue aplicando la <a href="#">configuración de retención de datos de Kinesis Data Streams</a> ).  Mínimo: -1  Máximo: 604 800

Parámetro	Obligatoria	Predeterminado	Notas
<a href="#">MaximumRetryAttempts</a>	N	-1	-1 significa infinito: se vuelven a intentar los registros que han producido error hasta que caduque el registro  Mínimo: -1  Máximo: 10 000
<a href="#">ParallelizationFactor</a>	N	1	Máximo: 10
<a href="#">StartingPosition</a>	Y	N/A	AT_TIMESTAMP, TRIM_HORIZON o LATEST
<a href="#">StartingPositionTimestamp</a>	N	N/A	Sólo es válido si StartingPosition se establece en AT_TIMESTAMP. El tiempo a partir del cual comenzar la lectura, en segundos de tiempo Unix
<a href="#">TumblingWindowInSeconds</a>	N	N/A	Mínimo: 0  Máximo: 900

## Uso del filtrado de eventos con una fuente de eventos de Kinesis

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo funciona el filtrado de eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para las fuentes de eventos de Kinesis.



## Temas

- [Conceptos básicos del filtrado de eventos de Kinesis](#)
- [Filtrado de registros agregados de Kinesis](#)

## Conceptos básicos del filtrado de eventos de Kinesis

Supongamos que un productor incluye datos con formato JSON en su flujo de datos de Kinesis. Un registro de ejemplo tendría el siguiente aspecto, con los datos JSON convertidos en una cadena codificada en Base64 en el campo `data`.

```
{
  "kinesis": {
    "kinesisSchemaVersion": "1.0",
    "partitionKey": "1",
    "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
    "data":
    "eyJJSZWNvcnR0dWliZXIiOiAiMDAwMSIsICJJaW1lU3RhbXAiOiAiZXl5eS1tbS1kZFRoaDptbTpcyIsICJSZXF1ZXN0",
    "approximateArrivalTimestamp": 1545084650.987
  },
  "eventSource": "aws:kinesis",
  "eventVersion": "1.0",
  "eventID":
  "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
  "eventName": "aws:kinesis:record",
  "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
  "awsRegion": "us-east-2",
  "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
}
```

Siempre que los datos que el productor incluya en el flujo sean JSON válidos, puede usar el filtrado de eventos para filtrar registros mediante la clave `data`. Supongamos que un productor incluye registros en su flujo de Kinesis en el siguiente formato JSON.

```
{
  "record": 12345,
  "order": {
    "type": "buy",
    "stock": "ANYCO",
    "quantity": 1000
  }
}
```

```
}
```

Para filtrar solo los registros en los que el tipo de pedido sea “comprar”, el objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "data": {
    "order": {
      "type": [ "buy" ]
    }
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/my-stream \
```

```
--filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type\n\" : [ \"buy\" ] } } }"]}]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type\n\" : [ \"buy\" ] } } }"]}]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:  
  Filters:  
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Para filtrar correctamente los eventos de orígenes de Kinesis, tanto el campo de datos como los criterios de filtro del campo de datos deben estar en un formato JSON válido. Si el formato JSON de alguno de los campos no es válido, Lambda elimina el mensaje o genera una excepción. En la siguiente tabla se resume el comportamiento específico:

Formato de los datos entrantes	Formato del patrón de filtro para las propiedades de datos	Acción resultante
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	No JSON	Lambda genera una excepción al crear o actualizar la asignación de origen de

Formato de los datos entrantes	Formato del patrón de filtro para las propiedades de datos	Acción resultante
		eventos. El formato JSON del patrón de filtro de las propiedades de datos debe ser válido.
No JSON	JSON válido	Lambda elimina el registro.
No JSON	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
No JSON	No JSON	Lambda genera una excepción al crear o actualizar la asignación de origen de eventos. El formato JSON del patrón de filtro de las propiedades de datos debe ser válido.

## Filtrado de registros agregados de Kinesis

Con Kinesis, puede agregar varios registros en un solo registro de Kinesis Data Streams para aumentar el rendimiento de sus datos. Lambda solo puede aplicar criterios de filtro a los registros agregados cuando se utiliza la [distribución mejorada](#) de Kinesis. El filtrado de registros agregados con Kinesis estándar no es compatible. Al utilizar la distribución mejorada, usted configura un consumidor de rendimiento dedicado de Kinesis para que actúe como desencadenador de la función de Lambda. A continuación, Lambda filtra los registros agregados y solo pasa los registros que cumplen los criterios de filtro.

Para obtener más información sobre la agregación de registros de Kinesis, consulte la sección [Agregación](#) de la página de conceptos clave de la Kinesis Producer Library (KPL). Para obtener más información sobre el uso de Lambda con la expansión mejorada de Kinesis, consulte [Aumento del rendimiento del procesamiento de transmisiones en tiempo real con la distribución mejorada de Amazon Kinesis Data Streams y AWS Lambda](#) en el blog de informática de AWS.

## Tutorial: Uso de Lambda con Kinesis Data Streams

En este tutorial, creará una función de Lambda para consumir eventos de un flujo de datos de Amazon Kinesis.

1. Una aplicación personalizada escribe los registros en el flujo.
2. AWS Lambda sondea el flujo y, cuando detecta registros nuevos en él, llama a la función de Lambda.
3. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.

### Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#), para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

#### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, `zip`) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.

## Creación del rol de ejecución

Cree el [rol de ejecución](#) que concederá a su función permiso para obtener acceso a los recursos de AWS.

Para crear un rol de ejecución

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Trusted entity (Entidad de confianza): AWS Lambda.
  - Permisos: AWSLambdaKinesisExecutionRole.
  - Nombre de rol: **lambda-kinesis-role**.

La política AWSLambdaKinesisExecutionRole tiene permisos que la función necesita para leer elementos de Kinesis y escribir registros a Registros de CloudWatch.

## Creación de la función

Cree una función de Lambda: que procese los mensajes de Kinesis. El código de función registra el ID del evento y los datos del evento del registro de Kinesis en Registros de CloudWatch.

En este tutorial, se utiliza el tiempo de ejecución de Node.js 18.x, pero también hemos proporcionado archivos de código de ejemplo en otros lenguajes de tiempo de ejecución. Puede seleccionar la pestaña del siguiente cuadro para ver el código del tiempo de ejecución que le interesa. El código JavaScript que usará en este paso está en el primer ejemplo que se muestra en la pestaña JavaScript.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de Kinesis con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
    }
}
```

```
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}
```

## Go

### SDK para Go V2

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
    if len(kinesisEvent.Records) == 0 {
        log.Printf("empty Kinesis event received")
    }
}
```



```
    return nil
}

for _, record := range kinesisEvent.Records {
    log.Printf("processed Kinesis event with EventId: %v", record.EventID)
    recordDataBytes := record.Kinesis.Data
    recordDataText := string(recordDataBytes)
    log.Printf("record data: %v", recordDataText)
    // TODO: Do interesting work based on the new data
}
log.Printf("successfully processed %v records", len(kinesisEvent.Records))
return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
```

```
public Void handleRequest(final KinesisEvent event, final Context context) {
    LambdaLogger logger = context.getLogger();
    if (event.getRecords().isEmpty()) {
        logger.log("Empty Kinesis Event received");
        return null;
    }
    for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
        try {
            logger.log("Processed Event with EventId: "+record.getEventID());
            String data = new String(record.getKinesis().getData().array());
            logger.log("Data:"+ data);
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex) {
            logger.log("An error occurred:"+ex.getMessage());
            throw ex;
        }
    }
    logger.log("Successfully processed:"+event.getRecords().size()+"
records");
    return null;
}
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
```

```

    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

## Uso de un evento de Kinesis con Lambda mediante TypeScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {

```

```
try {
    logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
    const recordData = await getRecordDataAsync(record.kinesis);
    logger.info(`Record Data: ${recordData}`);
    // TODO: Do interesting work based on the new data
} catch (err) {
    logger.error(`An error occurred ${err}`);
    throw err;
}
logger.info(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(
    payload: KinesisStreamRecordPayload
): Promise<string> {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
```

```
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
            marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de Kinesis con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
  here.
  return data
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Consumir un evento de Kinesis con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
{
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
```



```
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

## Cómo crear la función

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```
mkdir kinesis-tutorial
cd kinesis-tutorial
```

2. Copie el código de muestra de JavaScript en un nuevo archivo con el nombre `index.js`.
3. Cree un paquete de implementación.

```
zip function.zip index.js
```

4. Cree una función de Lambda con el comando `create-function`.

```
aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-kinesis-role
```

## Probar la función de Lambda

invoque la función de Lambda manualmente mediante el comando de la CLI de `invoke` AWS Lambda y un evento de Kinesis de muestra.

### Probar la función de Lambda

1. Copie el siguiente JSON en un archivo y guárdelo como `input.txt`.

```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
```

```

        "partitionKey": "1",
        "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
    },
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::111122223333:role/lambda-kinesis-
role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:111122223333:stream/
lambda-stream"
    }
]
}

```

- Utilice el comando `invoke` para enviar el evento a la función.

```

aws lambda invoke --function-name ProcessKinesisRecords \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt

```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

La respuesta se guardará en el archivo `output.txt`.

## Creación de un flujo de Kinesis

Utilice el comando `create-stream` para crear un flujo.

```

aws kinesis create-stream --stream-name lambda-stream --shard-count 1

```

Ejecute el siguiente comando `describe-stream` para obtener el ARN del flujo.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Debería ver los siguientes datos de salida:

```
{
  "StreamDescription": {
    "Shards": [
      {
        "ShardId": "shardId-000000000000",
        "HashKeyRange": {
          "StartingHashKey": "0",
          "EndingHashKey": "340282366920746074317682119384634633455"
        },
        "SequenceNumberRange": {
          "StartingSequenceNumber":
"49591073947768692513481539594623130411957558361251844610"
        }
      }
    ],
    "StreamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream",
    "StreamName": "lambda-stream",
    "StreamStatus": "ACTIVE",
    "RetentionPeriodHours": 24,
    "EnhancedMonitoring": [
      {
        "ShardLevelMetrics": []
      }
    ],
    "EncryptionType": "NONE",
    "KeyId": null,
    "StreamCreationTimestamp": 1544828156.0
  }
}
```

Utilice el ARN del flujo en el siguiente paso para asociar el flujo a la función de Lambda.

## Añadir un origen de eventos en AWS Lambda

Ejecute el siguiente comando `add-event-source` de la AWS CLI.

```
aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \  
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream \  
--event-source-arn arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

```
--batch-size 100 --starting-position LATEST
```

Tenga en cuenta el ID de mapeo para un uso posterior. Para obtener una lista de mapeos de orígenes de eventos, ejecute el comando `list-event-source-mappings`.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \  
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

En la respuesta, puede verificar que el valor de estado es `enabled`. Las asignaciones de orígenes de eventos se pueden deshabilitar para poner en pausa temporalmente el sondeo sin perder de registros.

## Prueba de la configuración

Para probar la asignación de orígenes de eventos, agregue los registros de eventos a su flujo de Kinesis. El valor de `--data` es una cadena que la CLI codifica en base64 antes de enviarlo a Kinesis. Puede ejecutar el mismo comando más de una vez para añadir varios registros al flujo.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \  
--data "Hello, this is a test."
```

Lambda utiliza el rol de ejecución para leer los registros desde el flujo. A continuación, se invoca la función de Lambda y se pasan lotes de registros. La función descodifica los datos de cada registro y los registra, enviando la salida a Registros de CloudWatch. Puede ver los registros en la [consola de CloudWatch](#).

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

## Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete (Eliminar).

## Para eliminar el flujo de Kinesis

1. Inicie sesión en la AWS Management Console y abra la consola de Kinesis en <https://console.aws.amazon.com/kinesis>.
2. Seleccione el flujo que ha creado.
3. Elija Actions (Acciones), Delete (Eliminar).
4. Introduzca **delete** en el campo de entrada de texto.
5. Elija Eliminar.

# Uso de Lambda con Amazon MQ

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

Amazon MQ es un servicio de agente de mensajes administrado para [Apache ActiveMQ](#) y [RabbitMQ](#). Un agente de mensajes permite que las aplicaciones de software y los componentes se comuniquen mediante varios lenguajes de programación, sistemas operativos y protocolos de mensajería formales a través de destinos de eventos de tema o de cola.

Amazon MQ también puede administrar instancias de Amazon Elastic Compute Cloud (Amazon EC2) en su nombre instalando agentes de ActiveMQ o RabbitMQ proporcionando diferentes topologías de red y otras necesidades de infraestructura.

Puede utilizar una función de Lambda para procesar registros de su agente de mensajes de Amazon MQ. Lambda invoca su función a través de una [asignación de orígenes de eventos](#), un recurso de Lambda que lee los mensajes de su agente e invoca la función [sincrónicamente](#).

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

La asignación de orígenes de eventos de Amazon MQ tiene las siguientes restricciones de configuración:

- **Simultaneidad:** las funciones de Lambda que utilizan una asignación de orígenes de eventos de Amazon MQ tienen una configuración de [simultaneidad](#) máxima predeterminada. Para ActiveMQ, el servicio Lambda limita la cantidad de entornos de ejecución simultánea a cinco por cada

asignación de orígenes de eventos de Amazon MQ. Para RabbitMQ, la cantidad de entornos de ejecución simultánea está limitada a uno por cada asignación de orígenes de eventos de Amazon MQ. Incluso si cambia la configuración de simultaneidad reservada o aprovisionada de la función, el servicio Lambda no ofrecerá más entornos de ejecución disponibles. Para solicitar un aumento de la simultaneidad máxima predeterminada de una sola asignación de orígenes de eventos de Amazon MQ, póngase en contacto con AWS Support con el UUID de la asignación de orígenes de eventos y la región. Como los aumentos se aplican al nivel de asignación de orígenes de eventos específico, no a nivel de cuenta o región, debe solicitar manualmente un aumento de escala para cada asignación de orígenes de eventos.

- Cuentas cruzadas: Lambda no admite el procesamiento de cuentas cruzadas. No se puede utilizar Lambda para procesar registros de un agente de mensajes de Amazon MQ que se encuentra en una Cuenta de AWS diferente.
- Autenticación: para ActiveMQ solo se admite ActiveMQ [SimpleAuthenticationPlugin](#). Para RabbitMQ, solo se admite el mecanismo de autenticación [PLAIN](#). Los usuarios deben utilizar AWS Secrets Manager para administrar sus credenciales. Para obtener más información acerca de la autenticación de ActiveMQ, consulte [Integración de agentes de ActiveMQ con LDAP](#) en la Guía para desarrolladores de Amazon MQ.
- Cuota de conexión: los agentes de cuota de conexión tienen un número máximo de conexiones permitidas por protocolo de nivel de cable. Esta cuota se basa en el tipo de instancia del broker. Para obtener más información, consulte la sección de [Agentes](#) de Cuotas en Amazon MQ en la Guía para desarrolladores de Amazon MQ.
- Conectividad: puede crear agentes en la nube privada virtual (VPC) pública o privada. En el caso de las VPC privadas, su función de Lambda necesita acceso a la VPC para recibir mensajes. Para obtener más información, consulte [the section called “Configuración del acceso de red”](#) más adelante en este tema.
- Destinos de eventos: solo se admiten los destinos de cola. Sin embargo, puede utilizar un tema virtual, que se comporta como un tema internamente mientras interactúa con Lambda como una cola. Para obtener más información, consulte [Destinos virtuales](#) en el sitio web de Apache ActiveMQ y [Virtual Hosts](#) en el sitio web RabbitMQ.
- Topología de red: para ActiveMQ, solo se admite una instancia única o agente en espera por asignación de orígenes de eventos. Para RabbitMQ, solo se admite una implementación de agente o clúster de una instancia única por asignación de orígenes de eventos. Los agentes de instancia única requieren un punto de conexión de conmutación por error. Para obtener más información acerca de estos modos de implementación de agente, consulte [Arquitectura de agente Active MQ](#) y [Arquitectura de agente de Rabbit MQ](#) en la Guía para desarrolladores de Amazon MQ.

- **Protocolos:** los protocolos compatibles dependen del tipo de integración de Amazon MQ.
  - Para las integraciones de ActiveMQ, Lambda consume mensajes mediante el protocolo OpenWire/Servicio de mensajes de Java (JMS). No se admiten otros protocolos para consumir mensajes. Dentro del protocolo JMS, solo [TextMessage](#) y [BytesMessage](#) son compatibles. Lambda también admite las propiedades personalizadas de JMS. Para obtener más información acerca del protocolo OpenWire, consulte [OpenWire](#) en el sitio web de Apache ActiveMQ.
  - Para las integraciones de RabbitMQ, Lambda consume mensajes utilizando el protocolo AMQP 0-9-1. No se admiten otros protocolos para consumir mensajes. Para obtener más información acerca de la implementación de RabbitMQ del protocolo AMQP 0-9-1, consulte la [Guía de referencia completa de AMQP 0-9-1](#) en el sitio web de RabbitMQ.

Lambda admite automáticamente las últimas versiones de ActiveMQ y RabbitMQ que admite Amazon MQ. Para obtener las últimas versiones compatibles, consulte [Notas de la versión de Amazon MQ](#) en la Guía para desarrolladores de Amazon MQ.

#### Note

De forma predeterminada, Amazon MQ tiene una ventana de mantenimiento semanal para los agentes. Durante esa ventana de tiempo, los corredores no están disponibles. Para los agentes sin espera, Lambda no puede procesar ningún mensaje durante esa ventana.

## Temas

- [Comprender el grupo de consumidores de Lambda para Amazon MQ](#)
- [Procesamiento de mensajes de Amazon MQ con Lambda](#)
- [Parámetros de asignación de orígenes de eventos](#)
- [Filtrar eventos de una fuente de eventos de Amazon MQ](#)
- [Solución de problemas de asignación de orígenes de eventos de Amazon MQ](#)

## Comprender el grupo de consumidores de Lambda para Amazon MQ

Para interactuar con Amazon MQ, Lambda crea un grupo de consumidores que puede leer de sus agentes de Amazon MQ. El grupo de consumidores se crea con el mismo ID que el UUID de asignación de orígenes de eventos.



Para los orígenes de eventos de Amazon MQ, Lambda agrupa los registros y los envía a su función en una sola carga. Para controlar el comportamiento, puede configurar el plazo de procesamiento por lotes y el tamaño del lote. Lambda extrae mensajes hasta que procesa el tamaño de carga máximo de 6 MB, el plazo de procesamiento por lotes vence o el número de registros alcanza el tamaño completo del lote. Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

El grupo de consumidores recupera los mensajes como un BLOB de bytes, los codifica con base64 en una sola carga JSON y luego invoca la función. Si su función devuelve un error para cualquiera de los mensajes de un lote, Lambda reintenta todo el lote de mensajes hasta que el procesamiento sea correcto o los mensajes caduquen.

### Note

Si bien las funciones de Lambda suelen tener un límite de tiempo de espera máximo de 15 minutos, las asignaciones de orígenes de eventos para Amazon MSK, Apache Kafka autoadministrado, Amazon DocumentDB y Amazon MQ para ActiveMQ y RabbitMQ solo admiten funciones con límites de tiempo de espera máximos de 14 minutos. Esta restricción garantiza que la asignación de orígenes de eventos pueda gestionar correctamente los errores y reintentos de las funciones.

Puede supervisar el uso de concurrencia de una función determinada utilizando la métrica `ConcurrentExecutions` en Amazon CloudWatch. Para obtener más información acerca de la simultaneidad, consulte [the section called "Configuración de la simultaneidad reservada"](#).

Example Eventos de registro de Amazon MQ

ActiveMQ

```
{
  "eventSource": "aws:mq",
  "eventSourceArn": "arn:aws:mq:us-west-2:111122223333:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "messages": [
    {
      "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
      "messageType": "jms/text-message",
      "deliveryMode": 1,
    }
  ]
}
```

```
"replyTo": null,
"type": null,
"expiration": "60000",
"priority": 1,
"correlationId": "myJMScoID",
"redelivered": false,
"destination": {
  "physicalName": "testQueue"
},
"data": "QUJD0kFBQUE=",
"timestamp": 1598827811958,
"brokerInTime": 1598827811958,
"brokerOutTime": 1598827811959,
"properties": {
  "index": "1",
  "doAlarm": "false",
  "myCustomProperty": "value"
}
},
{
  "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
  "messageType": "jms/bytes-message",
  "deliveryMode": 1,
  "replyTo": null,
  "type": null,
  "expiration": "60000",
  "priority": 2,
  "correlationId": "myJMScoID1",
  "redelivered": false,
  "destination": {
    "physicalName": "testQueue"
  },
  "data": "LQaGQ82S48k=",
  "timestamp": 1598827811958,
  "brokerInTime": 1598827811958,
  "brokerOutTime": 1598827811959,
  "properties": {
    "index": "1",
    "doAlarm": "false",
    "myCustomProperty": "value"
  }
}
]
```

```
}
```

## RabbitMQ

```
{
  "eventSource": "aws:rmq",
  "eventSourceArn": "arn:aws:mq:us-
west-2:111122223333:broker:pizzaBroker:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "pizzaQueue::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                50
              ]
            },
            "numberInHeader": 10
          },
          "deliveryMode": 1,
          "priority": 34,
          "correlationId": null,
          "replyTo": null,
          "expiration": "60000",

```

```
    "messageId": null,
    "timestamp": "Jan 1, 1970, 12:33:41 AM",
    "type": null,
    "userId": "AIDACKCEVSQ6C2EXAMPLE",
    "appId": null,
    "clusterId": null,
    "bodySize": 80
  },
  "redelivered": false,
  "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}
]
}
```

### Note

En el ejemplo de RabbitMQ, `pizzaQueue` es el nombre de la cola de RabbitMQ y `/` es el nombre del host virtual. Al recibir mensajes, el origen de eventos muestra los mensajes `enpizzaQueue:./`.

## Procesamiento de mensajes de Amazon MQ con Lambda

### Temas

- [Configuración del acceso de red](#)
- [Creación de la asignación de orígenes de eventos](#)

### Configuración del acceso de red

Para que Lambda tenga acceso completo a su agente a través de su asignación de orígenes de eventos, el agente debe utilizar un punto de conexión público (dirección IP pública) o bien debe proporcionar acceso a la instancia de Amazon VPC en la que creó el agente.

De forma predeterminada, al crear un agente de Amazon MQ, la marca `PubliclyAccessible` está establecida en falso. Para que su agente reciba una dirección IP pública, debe establecer la marca `PubliclyAccessible` en verdadero.

La práctica recomendada para utilizar Amazon MQ con Lambda consiste en utilizar [puntos de conexión de VPC](#) de AWS PrivateLink y permitir que la función de Lambda acceda a la VPC de su agente. Implemente un punto de conexión para Lambda y, solo para ActiveMQ, un punto de conexión para AWS Security Token Service (AWS STS). Si el agente utiliza autenticación, implemente también un punto de conexión de VPC para AWS Secrets Manager. Para obtener más información, consulte [the section called “Configuración de los puntos de conexión de VPC de Amazon”](#).

Configure una puerta de enlace de NAT en cada subred pública de la VPC que contenga su agente de Amazon MQ. Para obtener más información, consulte [the section called “Acceso a Internet para funciones de VPC”](#).

Al crear una asignación de orígenes de eventos para un agente de Amazon MQ, Lambda comprueba si las interfaces de red elásticas (ENI) ya están presentes en las subredes y los grupos de seguridad de la VPC de su agente. Si Lambda encuentra ENI existentes, intenta reutilizarlos. De lo contrario, Lambda crea nuevos ENI para conectarse al origen de eventos e invocar la función.

#### Note

Las funciones de Lambda siempre se ejecutan dentro de VPC propiedad del servicio de Lambda. El servicio mantiene estas VPC de forma automática, y no son visibles para los clientes. También puede conectar su función a una Amazon VPC. En cualquier caso, la configuración de VPC de la función no afecta la asignación de orígenes de eventos. Solo la configuración de la VPC del origen de eventos determina cómo Lambda se conecta al origen de eventos.

## Configuración de reglas de grupos de seguridad

Configure los grupos de seguridad para la Amazon VPC que contiene su Amazon MQ.

#### Note

De forma predeterminada, Amazon MQ para ActiveMQ utiliza el puerto 61617 y Amazon MQ para RabbitMQ utiliza el puerto 5671.

- Reglas de entrada: permiten todo el tráfico en el puerto del agente para el grupo de seguridad asociado con el origen de eventos desde dentro del propio grupo de seguridad.

- Reglas de salida: permiten todo el tráfico en el puerto 443 para todos los destinos. Permita todo el tráfico en el puerto del agente desde dentro de su propio grupo de seguridad.
- Reglas de entrada del punto de conexión de VPC de Amazon: para el grupo de seguridad asociado al punto de conexión de VPC de Amazon, permita el tráfico entrante en el puerto 443 desde el grupo de seguridad del agente

## Configuración de los puntos de conexión de VPC de Amazon

Cuando utiliza puntos de conexión de VPC de Amazon, AWS direcciona las llamadas a la API para invocar una función mediante la interfaz de red elástica (ENI) del punto de conexión. La entidad principal del servicio de Lambda debe llamar a `lambda:InvokeFunction` para cualquier función que utilice esos ENI. Además, en el caso de Amazon MQ para ActiveMQ, la entidad principal del servicio de Lambda debe ejecutar `sts:AssumeRole` en roles que utilicen los ENI.

De forma predeterminada, los puntos de conexión de VPC de Amazon tienen políticas de IAM abiertas que permiten un amplio acceso a los recursos. Para usar Amazon MQ con Lambda en producción, puede restringir estas políticas para permitir que solo entidades principales específicas accedan únicamente a funciones y roles específicos.

En el siguiente ejemplo de políticas de puntos de conexión de VPC, se muestra cómo conceder el acceso necesario a AWS STS y los puntos de conexión de Lambda.

### Example Política de puntos de conexión: punto de conexión de AWS STS (solo ActiveMQ)

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "arn:aws::iam::123456789012:role/my-role"
    }
  ]
}
```

## Example Política de puntos de conexión: punto de conexión de Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "arn:aws::lambda:us-west-2:123456789012:function:my-function"
    }
  ]
}
```

Si su agente de Amazon MQ utiliza la autenticación, también puede restringir la política del punto de conexión para el punto de conexión de Secrets Manager. Para llamar a la API de Secrets Manager, Lambda usa su rol de función, no la entidad principal de servicio de Lambda. En el siguiente ejemplo, se muestra una política del punto de conexión de Secrets Manager.

## Example Política de punto de conexión de VPC: punto de conexión de Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws::iam::123456789012:role/my-role"
        ]
      },
      "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-secret"
    }
  ]
}
```

**⚠ Warning**

Restringir las políticas de puntos de conexión para que solo permitan las llamadas a la API que se originen en su organización impide que la asignación de orígenes de eventos funcione de forma correcta.

## Creación de la asignación de orígenes de eventos

Cree una [asignación de orígenes de eventos](#) para indicar que envíe registros de un agente de Amazon MQ a una función de Lambda. Puede crear varias asignaciones de orígenes de eventos para procesar los mismos datos con distintas funciones o para procesar elementos de varios orígenes con una sola función.

Para configurar la función para leer de Amazon MQ, agregue los permisos requeridos y cree un desencadenador de MQ en la consola de Lambda.

Para leer registros desde un agente de Amazon MQ, la función de Lambda necesita los siguientes permisos. Para conceder a Lambda permiso para interactuar con el agente de Amazon MQ y sus recursos subyacentes, agregue instrucciones de permisos a la función del [rol de ejecución](#):

- [mq:DescribeBroker](#)
- [secretsmanager:GetSecretValue](#)
- [ec2:CreateNetworkInterface](#)
- [ec2>DeleteNetworkInterface](#)
- [ec2:DescribeNetworkInterfaces](#)
- [ec2:DescribeSecurityGroups](#)
- [ec2:DescribeSubnets](#)
- [ec2:DescribeVpcs](#)
- [logs:CreateLogGroup](#)
- [logs:CreateLogStream](#)
- [logs:PutLogEvents](#)

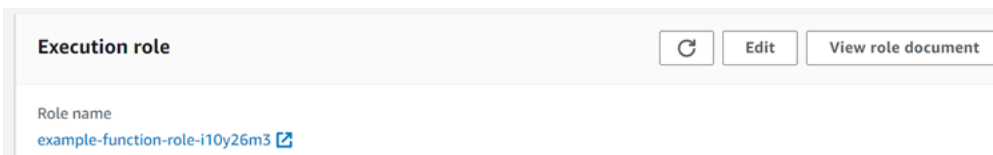


**Note**

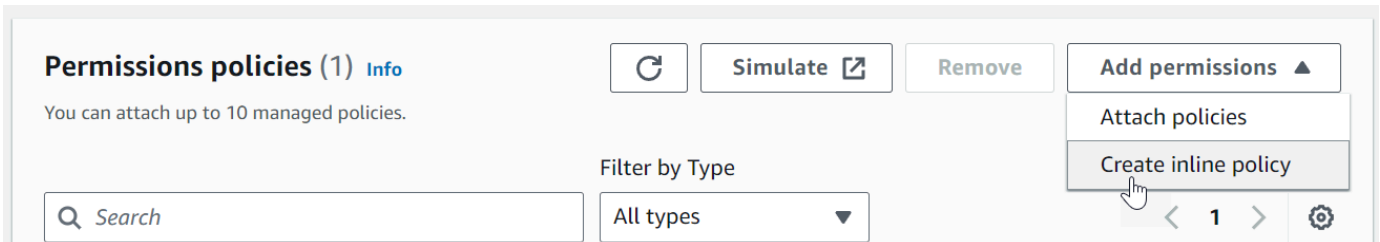
Cuando utilice una clave administrada por el cliente cifrada, agregue también el permiso [kms:Decrypt](#).

## Cómo agregar permisos y crear un desencadenador

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija la pestaña Configuración y, a continuación, elija Permisos.
4. En Nombre del rol, elija el enlace al rol de ejecución. Este enlace abre el rol en la consola de IAM.



5. Seleccione Agregar permisos y, a continuación, Crear política insertada.



6. En Editor de políticas, elija JSON. Escriba la siguiente política. La función necesita estos permisos de lectura desde un agente de Amazon MQ.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mq:DescribeBroker",
        "secretsmanager:GetSecretValue",
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",

```

```

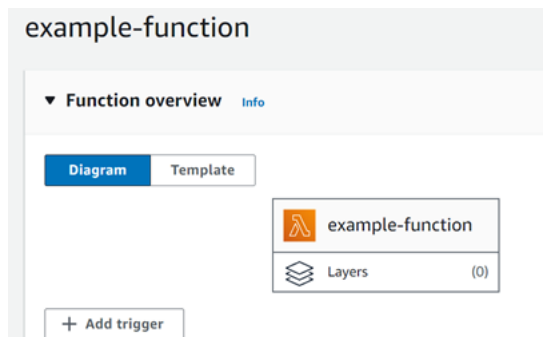
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents"
  ],
  "Resource": "*"
}
]
}

```

### Note

Cuando utilice una clave cifrada y administrada por el cliente, agregue también el permiso `kms:Decrypt`.

7. Elija Siguiente. Introduzca un nombre de política y, a continuación, elija Crear política.
8. Regrese a la función en la consola de Lambda. En Descripción general de la función, elija Agregar desencadenador.



9. Elija el tipo de desencadenador MQ.
10. Configure las opciones requeridas y luego elija Agregar.

Lambda admite las siguientes opciones para los orígenes de eventos de Amazon MQ.

- Agente de MQ: seleccione un agente de Amazon MQ.
- Tamaño del lote; establezca el número máximo de mensajes que se recuperarán en un solo lote.
- Nombre de la cola: escriba la cola de Amazon MQ que se va a consumir.
- Configuración del acceso al origen: introduzca la información del host virtual y el secreto de Secrets Manager que almacena las credenciales del agente.

- Activar desencadenador: desactive el desencadenador para detener el procesamiento de registros.

Para habilitar o desactivar el desencadenador (o eliminarlo), elija el desencadenador de MQ en el diseñador. Para volver a configurar el desencadenador, utilice las operaciones de API de asignación de origen de eventos.

## Parámetros de asignación de orígenes de eventos

Todos los tipos de fuente de eventos Lambda comparten las mismas operaciones

[CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Amazon MQ y RabbitMQ.

Parámetro	Obligatoria	Predeterminado	Notas
BatchSize	N	100	Máximo: 10 000
Habilitado	N	true	Ninguno
FunctionName	Y	N/A	Ninguno
FilterCriteria	N	N/A	<a href="#">Controle qué eventos envía Lambda a la función</a>
MaximumBatchingWindowInSeconds	N	500 ms	<a href="#">Comportamiento de procesamiento por lotes</a>
Queues	N	N/A	Nombre de la cola de destino del agente de Amazon MQ que se va a consumir.
SourceAccessConfigurations	N	N/A	Para ActiveMQ, credenciales BASIC_AUTH. Para RabbitMQ, puede contener tanto credencia

Parámetro	Obligatoria	Predeterminado	Notas
			les BASIC_AUTH como información de VIRTUAL_HOST.

## Filtrar eventos de una fuente de eventos de Amazon MQ

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo funciona el filtrado de eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para las fuentes de eventos de Amazon MQ.

### Temas

- [Conceptos básicos de filtrado de eventos de Amazon MQ](#)

## Conceptos básicos de filtrado de eventos de Amazon MQ

Supongamos que su cola de mensajes de Amazon MQ contiene mensajes en formato JSON válido o como cadenas simples. Un registro de ejemplo tendría el siguiente aspecto, con los datos convertidos en una cadena codificada en Base64 en el campo `data`.

### ActiveMQ

```
{
  "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
  "messageType": "jms/text-message",
  "deliveryMode": 1,
  "replyTo": null,
  "type": null,
  "expiration": "60000",
  "priority": 1,
  "correlationId": "myJMScoID",
  "redelivered": false,
  "destination": {
    "physicalName": "testQueue"
  },
  "data": "QUJD0kFBQUE="
}
```

```
"timestamp": 1598827811958,
"brokerInTime": 1598827811958,
"brokerOutTime": 1598827811959,
"properties": {
  "index": "1",
  "doAlarm": "false",
  "myCustomProperty": "value"
}
}
```

## RabbitMQ

```
{
  "basicProperties": {
    "contentType": "text/plain",
    "contentEncoding": null,
    "headers": {
      "header1": {
        "bytes": [
          118,
          97,
          108,
          117,
          101,
          49
        ]
      },
      "header2": {
        "bytes": [
          118,
          97,
          108,
          117,
          101,
          50
        ]
      },
      "numberInHeader": 10
    },
    "deliveryMode": 1,
    "priority": 34,
    "correlationId": null,
  }
}
```

```

    "replyTo": null,
    "expiration": "60000",
    "messageId": null,
    "timestamp": "Jan 1, 1970, 12:33:41 AM",
    "type": null,
    "userId": "AIDACKCEVSQ6C2EXAMPLE",
    "appId": null,
    "clusterId": null,
    "bodySize": 80
  },
  "redelivered": false,
  "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}

```

Tanto para los agentes de Active MQ como los de Rabbit MQ, puede utilizar el filtrado de eventos para filtrar los registros mediante la clave data. Supongamos que su cola de Amazon MQ contiene mensajes en el siguiente formato JSON.

```

{
  "timeout": 0,
  "IPAddress": "203.0.113.254"
}

```

Para filtrar solo los registros en los que el campo timeout sea mayor que 0, el objeto `FilterCriteria` sería el siguiente.

```

{
  "Filters": [
    {
      "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\" : [ \">\",
0] ] } ] } }"
    }
  ]
}

```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```

{
  "data": {
    "timeout": [ { "numeric": [ ">", 0 ] } ]
  }
}

```

```
}
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-  
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :  
[ { \"numeric\": [ \">\", 0 ] } ] } }"]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :  
[ { \"numeric\": [ \">\", 0 ] } ] } }"]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Con Amazon MQ, también puede filtrar los registros en los que el mensaje sea una cadena simple. Supongamos que desea procesar solo los registros en los que el mensaje comienza por "Resultado:". El objeto `FilterCriteria` tendría el siguiente aspecto.

```
{
  "Filters": [
    {
      "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "data": [
    {
      "prefix": "Result: "
    }
  ]
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

### AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-  
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
```



```
--filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"]}]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"]}]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

Los mensajes de Amazon MQ deben ser cadenas codificadas en UTF-8, cadenas simples o en formato JSON. Esto se debe a que Lambda decodifica las matrices de bytes de Amazon MQ en UTF-8 antes de aplicar los criterios de filtro. Si los mensajes utilizan otra codificación, como UTF-16 o ASCII, o el formato del mensaje no coincide con el formato de `FilterCriteria`, Lambda solo procesa los filtros de metadatos. En la siguiente tabla se resume el comportamiento específico:

Formato del mensaje entrante	Formato del patrón de filtro para las propiedades del mensaje	Acción resultante
Cadena sin formato	Cadena sin formato	Lambda filtra en función de los criterios de filtro.
Cadena sin formato	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.

Formato del mensaje entrante	Formato del patrón de filtro para las propiedades del mensaje	Acción resultante
Cadena sin formato	JSON válido	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Cadena sin formato	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.
Cadena no codificada con UTF-8	JSON, cadena sin formato o sin patrón	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.

## Solución de problemas de asignación de orígenes de eventos de Amazon MQ

Cuando una función de Lambda encuentra un error irrecuperable, el consumidor de Amazon MQ dejará de procesar registros. Cualquier otro consumidor puede continuar procesando, siempre que no encuentre el mismo error. Para determinar la causa potencial de un consumidor detenido, marque el campo `StateTransitionReason` en los detalles de devolución de su `EventSourceMapping` para obtener uno de los siguientes códigos:

**ESM\_CONFIG\_NOT\_VALID**

La configuración de asignación de orígenes de eventos no es válida.

**EVENT\_SOURCE\_AUTHN\_ERROR**

Lambda ha producido un error al autenticar el origen del evento.

**EVENT\_SOURCE\_AUTHZ\_ERROR**

Lambda no tiene los permisos necesarios para acceder al origen de eventos.

**FUNCTION\_CONFIG\_NOT\_VALID**

La configuración de la función no es válida.

Los registros también quedan sin procesar si Lambda los elimina debido a su tamaño. El límite de tamaño para los registros de Lambda es de 6 MB. Para volver a entregar mensajes en caso de error de función, puede utilizar una política de cola de mensajes fallidos (DLQ). Para obtener más información, consulte [Reentrega de mensajes y administración de DLQ](#) en el sitio web de Apache ActiveMQ y [Guía de la fiabilidad](#) en el sitio web de RabbitMQ.

**Note**

Lambda no admite políticas de reenvío personalizadas. En su lugar, Lambda utiliza una política con los valores predeterminados de la página de [Política de reentrega](#) en el sitio web de Apache ActiveMQ, con `maximumRedeliveries` establecido en 6.

# Uso de Lambda con Amazon MSK

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

[Amazon Managed Streaming para Apache Kafka \(Amazon MSK\)](#) es un servicio completamente administrado que le permite crear y ejecutar aplicaciones que utilizan Apache Kafka para procesar datos de transmisión. Amazon MSK simplifica la configuración, el escalado y la administración de clústeres que ejecutan Kafka. Amazon MSK también facilita la configuración de la aplicación para varias zonas de disponibilidad y para la seguridad con AWS Identity and Access Management (IAM). Amazon MSK es compatible con múltiples versiones de código abierto de Kafka.

Amazon MSK como origen de eventos funciona de manera similar al uso de Amazon Simple Queue Service (Amazon SQS) o Amazon Kinesis. Lambda sondea internamente nuevos mensajes de la fuente del evento y luego invoca sincrónicamente la función de Lambda objetivo. Lambda lee los mensajes en lotes y los proporciona a su función como carga de eventos. El tamaño máximo del lote se puede configurar (el valor predeterminado son 100 mensajes). Para obtener más información, consulte [Comportamiento de procesamiento por lotes](#).

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

Para ver un ejemplo de cómo configurar Amazon MSK como origen de eventos, consulte [Uso de Amazon MSK como origen de eventos AWS Lambda](#) en el Blog de informática de AWS. Para obtener un tutorial completo, consulte [Amazon MSK Lambda Integration](#) (Integración de Amazon MSK y Lambda) en Amazon MSK Labs.

## Temas

- [Evento de ejemplo](#)
- [Configuración de orígenes de eventos de Amazon MSK para Lambda](#)
- [Procesamiento de mensajes de Amazon MSK con Lambda](#)
- [Filtrado de eventos con un origen de eventos de Amazon MSK](#)
- [Capturar lotes descartados para un origen de eventos de Amazon MSK](#)
- [Tutorial: Uso de una asignación de orígenes de eventos de Amazon MSK para invocar una función de Lambda](#)

## Evento de ejemplo

Lambda envía el lote de mensajes en el parámetro de evento cuando invoca su función. La carga de eventos contiene una matriz de mensajes. Cada elemento de matriz contiene detalles del tema y el identificador de partición de Amazon MSK, junto con una marca de hora y un mensaje codificado en base64.

```
{
  "eventSource": "aws:kafka",
  "eventSourceArn": "arn:aws:kafka:sa-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
  "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-east-1.amazonaws.com:9092",
  "records": {
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXYZ1234==",
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "headers": [
          {
            "headerKey": [
              104,
              101,
              97,
```

```
        100,  
        101,  
        114,  
        86,  
        97,  
        108,  
        117,  
        101  
    ]  
  }  
] }  
] }  
] }  
}
```

## Configuración de orígenes de eventos de Amazon MSK para Lambda

Antes de crear una asignación de orígenes de eventos para el clúster de Amazon MSK, debe asegurarse de que tanto el clúster como la VPC en la que se encuentra estén correctamente configurados. También debe asegurarse de que el [rol de ejecución](#) de la función de Lambda tenga los permisos de IAM necesarios.

Siga las instrucciones en las siguientes secciones para configurar el clúster de Amazon MSK, la VPC y la función de Lambda. Para obtener información sobre cómo crear la asignación de orígenes de eventos, consulte [the section called “Agregar Amazon MSK como origen de eventos”](#).

### Temas

- [Autenticación de clústeres de MSK](#)
- [Administración del acceso de la API y los permisos](#)
- [Errores de autenticación y autorización](#)
- [Configuración de red](#)

## Autenticación de clústeres de MSK

Lambda necesita permiso para acceder al clúster de Amazon MSK, recuperar registros y llevar a cabo otras tareas. Amazon MSK admite varias opciones para controlar el acceso de los clientes al clúster de MSK.

## Opciones de acceso al clúster

- [Acceso sin autenticar](#)
- [Autenticación SASL/SCRAM](#)
- [Autenticación basada en roles de IAM](#)
- [Autenticación TLS mutua](#)
- [Configuración del secreto de mTLS](#)
- [Cómo Lambda elige un agente de arranque](#)

### Acceso sin autenticar

Si ningún cliente accede al clúster a través de Internet, puede utilizar el acceso no autenticado.

### Autenticación SASL/SCRAM

Amazon MSK admite autenticación simple y autenticación de capa de seguridad/mecanismo de autenticación de respuesta por desafío saltado (SASL/SCRAM) con cifrado de seguridad de la capa de transporte (TLS). Para que Lambda se conecte al clúster, las credenciales de autenticación (nombre de usuario y contraseña) se almacenan en un secreto de AWS Secrets Manager.

Para obtener más información sobre Secrets Manager, consulte [Autenticación de usuario y contraseña con AWS Secrets Manager](#) (en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka).

Amazon MSK no admite la autenticación SASL/PLAIN.

### Autenticación basada en roles de IAM

Puede utilizar IAM para autenticar la identidad de los clientes que se conectan al clúster de MSK. Si la autenticación de IAM está activa en el clúster de MSK y no proporciona un secreto para la autenticación, Lambda utilizará de forma automática la autenticación de IAM. Para crear e implementar políticas basadas en roles o usuarios, utilice la API o la consola de IAM. Para obtener más información, consulte [IAM access control](#) (Control de acceso de IAM) en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka.

Para permitir que Lambda se conecte al clúster de MSK, lea registros y lleve a cabo otras acciones necesarias, agregue los siguientes permisos al [rol de ejecución](#) de la función.

```
{  
  "Version": "2012-10-17",
```

```

    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "kafka-cluster:Connect",
          "kafka-cluster:DescribeGroup",
          "kafka-cluster:AlterGroup",
          "kafka-cluster:DescribeTopic",
          "kafka-cluster:ReadData",
          "kafka-cluster:DescribeClusterDynamicConfiguration"
        ],
        "Resource": [
          "arn:aws:kafka:region:account-id:cluster/cluster-name/cluster-uuid",
          "arn:aws:kafka:region:account-id:topic/cluster-name/cluster-uuid/topic-
name",
          "arn:aws:kafka:region:account-id:group/cluster-name/cluster-
uuid/consumer-group-id"
        ]
      }
    ]
  }
}

```

Puede asignar estos permisos a un clúster, un tema y un grupo específicos. Para obtener más información, consulte [Acciones de Amazon MSK para Kafka](#) en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka.

### Autenticación TLS mutua

TLS mutua (mTLS) proporciona autenticación bidireccional entre el cliente y el servidor. El cliente envía un certificado al servidor para que el servidor verifique el cliente, mientras que el servidor envía un certificado al cliente para que el cliente verifique el servidor.

En el caso de Amazon MSK, Lambda actúa como cliente. Puede configurar un certificado de cliente (como secreto en Secrets Manager) para autenticar a Lambda con los agentes del clúster de MSK. El certificado de cliente debe estar firmado por una entidad de certificación en el almacén de confianza del servidor. El clúster de MSK envía un certificado de servidor a Lambda para autenticar a los agentes con Lambda. El certificado de servidor debe estar firmado por una entidad de certificación que esté en el almacén de confianza de AWS.

Para obtener instrucciones sobre cómo generar un certificado de cliente, consulte [Introducing mutual TLS authentication for Amazon MSK as an event source](#) (Presentación de la autenticación con TLS mutua para Amazon MSK como origen de eventos).



Amazon MSK no admite certificados de servidor autofirmados porque todos los agentes de Amazon MSK utilizan [certificados públicos](#) firmados por [entidades de certificación de Amazon Trust Services](#), en los que Lambda confía de forma predeterminada.

Para obtener más información sobre mTLS para Amazon MSK, consulte [Mutual TLS Authentication](#) (Autenticación con TLS mutua) en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka.

### Configuración del secreto de mTLS

El secreto CLIENT\_CERTIFICATE\_TLS\_AUTH requiere un campo de certificado y un campo de clave privada. Para una clave privada cifrada, el secreto requiere una contraseña de clave privada. El certificado y la clave privada deben estar en formato PEM.

#### Note

Lambda admite los algoritmos de cifrado de claves privadas [PBES1](#) (pero no PBES2).

El campo de certificado debe contener una lista de certificados y debe comenzar por el certificado de cliente, seguido de cualquier certificado intermedio, y finalizar con el certificado raíz. Cada certificado debe comenzar en una nueva línea con la siguiente estructura:

```
-----BEGIN CERTIFICATE-----  
    <certificate contents>  
-----END CERTIFICATE-----
```

Secrets Manager admite secretos de hasta 65 536 bytes, que supone suficiente espacio para cadenas de certificados largas.

El formato de la clave privada debe ser [PKCS #8](#), con la siguiente estructura:

```
-----BEGIN PRIVATE KEY-----  
    <private key contents>  
-----END PRIVATE KEY-----
```

Para una clave privada cifrada, utilice la siguiente estructura:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
```

```
<private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

El siguiente ejemplo muestra el contenido de un secreto para la autenticación de mTLS mediante una clave privada cifrada. Para una clave privada cifrada, incluya la contraseña de la clave privada en el secreto.

```
{
  "privateKeyPassword": "testpassword",
  "certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPim040s1cRqtFHxoa10QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFGjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMg0SA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
  "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBGkqhkiG9w0BBQ0wSDANBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzg09IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

## Cómo Lambda elige un agente de arranque

Lambda elige un [agente de arranque](#) en función de los métodos de autenticación disponibles en el clúster y de si proporciona un secreto para la autenticación. Si proporciona un secreto para mTLS o SASL/SCRAM, Lambda elige de forma automática ese método de autenticación. Si no proporciona ningún secreto, Lambda selecciona el método de autenticación más seguro que esté activo en el clúster. El siguiente es el orden de prioridad en el que Lambda selecciona un agente, de la autenticación más segura a la menos segura:

- mTLS (secreto proporcionado para mTLS)
- SASL/SCRAM (secreto proporcionado para SASL/SCRAM)
- SASL IAM (no se proporciona secreto y la autenticación de IAM está activa)

- TLS no autenticada (no se proporciona secreto y la autenticación de IAM no está activa)
- Texto sin formato (no se proporciona secreto y tanto la autenticación de IAM como la TLS no autenticada no están activas)

#### Note

Si Lambda no puede conectarse al tipo de agente más seguro, no intentará conectarse a un tipo de agente diferente (menos seguro). Si quiere que Lambda elija un tipo de agente más débil, desactive todos los métodos de autenticación más seguros del clúster.

## Administración del acceso de la API y los permisos

Además de acceder al clúster de Amazon MSK, la función necesita permisos para llevar a cabo varias acciones de la API de Amazon MSK. Los permisos se agregan al rol de ejecución de la función. Si los usuarios tienen que acceder a cualquier acción de la API de Amazon MSK, agregue los permisos necesarios a la política de identidad para el usuario o rol.

Puede agregar cada uno de los siguientes permisos a su rol de ejecución de forma manual. Como alternativa, puede adjuntar la política administrada por AWS [AWSLambdaMSKExecutionRole](#) a su rol de ejecución. La política `AWSLambdaMSKExecutionRole` contiene todas las acciones de API y los permisos de VPC necesarios que se enumeran a continuación.

### Permisos de rol de ejecución de la función de Lambda necesarios

Para crear y almacenar registros en un grupo de registros en Registros de Amazon CloudWatch, la función de Lambda debe tener los siguientes permisos en su rol de ejecución:

- [logs:CreateLogGroup](#)
- [logs:CreateLogStream](#)
- [logs:PutLogEvents](#)

Para que Lambda acceda al clúster de Amazon MSK en su nombre, la función de Lambda debe tener los siguientes permisos en su rol de ejecución:

- [kafka:DescribeCluster](#)
- [kafka:DescribeClusterV2](#)

- [kafka:GetBootstrapBrokers](#)
- [kafka:DescribeVpcConnection](#): solo se requiere para las [asignaciones de orígenes de eventos entre cuentas](#).
- [kafka:ListVpcConnections](#): no es obligatorio en el rol de ejecución, pero sí es obligatorio para una entidad principal de IAM que esté creando una [asignación de orígenes de eventos entre cuentas](#).

Solo tiene que agregar el permiso `kafka:DescribeCluster` o el `kafka:DescribeClusterV2`. En el caso de los clústeres de MSK aprovisionados, cualquiera de los dos permisos funciona. Para los clústeres de MSK sin servidor, debe utilizar `kafka:DescribeClusterV2`.

#### Note

Lambda planea eliminar en su momento el permiso `kafka:DescribeCluster` de la política administrada asociada `AWSLambdaMSKExecutionRole`. Si utiliza esta política, debería migrar cualquier aplicación con `kafka:DescribeCluster` para utilizar `kafka:DescribeClusterV2` en su lugar.

## Permisos de VPC

Si solo los usuarios dentro de una VPC pueden acceder a su clúster de Amazon MSK, su función de Lambda debe tener permiso para acceder a sus recursos de Amazon VPC. Estos recursos incluyen su VPC, subredes, grupos de seguridad e interfaces de red. Para acceder a estos recursos, el rol de ejecución de la función debe tener los siguientes permisos. Estos permisos están incluidos en la política administrada por AWS [AWSLambdaMSKExecutionRole](#).

- [ec2:CreateNetworkInterface](#)
- [ec2:DescribeNetworkInterfaces](#)
- [ec2:DescribeVpcs](#)
- [ec2>DeleteNetworkInterface](#)
- [ec2:DescribeSubnets](#)
- [ec2:DescribeSecurityGroups](#)

## Permisos de función de Lambda opcionales

Es posible que la función de Lambda también necesite permisos para:

- Acceda a su secreto de SCRAM, mediante la autenticación SASL/SCRAM.
- Describir el secreto de Secrets Manager.
- Acceder a su clave administrada por el cliente de AWS Key Management Service (AWS KMS).
- Envíe los registros de las invocaciones fallidas a un destino.

## Secrets Manager y permisos de AWS KMS

En función del tipo de control de acceso que configure para los agentes de Amazon MSK, es posible que la función de Lambda necesite permiso para acceder a su secreto de SCRAM (si utiliza autenticación SASL/SCRAM) o al secreto de Secrets Manager para descifrar su clave administrada por el cliente de AWS KMS. Para acceder a estos recursos, el rol de ejecución de la función debe tener los siguientes permisos:

- [kafka:ListScramSecrets](#)
- [secretsmanager:GetSecretValue](#)
- [kms:Decrypt](#)

## Adición de permisos a su rol de ejecución

Siga estos pasos para agregar la política administrada por AWS [AWSLambdaMSKExecutionRole](#) a su rol de ejecución mediante la consola de IAM.

## Cómo agregar una política administrada de AWS

1. Abra la página [Políticas \(Políticas\)](#) en la consola de IAM.
2. En el cuadro de búsqueda, escriba el nombre de la política (`AWSLambdaMSKExecutionRole`).
3. Seleccione la política de la lista y, a continuación, elija Acciones de política, Adjuntar.
4. En la página Attach policy (Asociar política), seleccione su rol de ejecución en la lista y, a continuación, elija Attach policy (Asociar política).

## Concesión de acceso a los usuarios con una política de IAM

De forma predeterminada, los usuarios y roles no tienen permiso para llevar a cabo operaciones de API de Amazon MSK. Para conceder acceso a los usuarios de su organización o cuenta, puede agregar o actualizar la política basada en identidades. Para obtener más información, consulte

[Ejemplos de políticas basadas en identidades de Amazon MSK](#) en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka.

## Errores de autenticación y autorización

Si falta alguno de los permisos necesarios para consumir datos del clúster de Amazon MSK, Lambda muestra uno de los siguientes mensajes de error en la asignación del origen de eventos en `LastProcessingResult`.

### Mensajes de error

- [Error de autorización del clúster a Lambda](#)
- [Error de autenticación de SASL](#)
- [El servidor no pudo autenticar Lambda](#)
- [La clave privada o el certificado proporcionados no son válidos](#)

### Error de autorización del clúster a Lambda

Para SASL/SCRAM o mTLS, este error indica que el usuario proporcionado no tiene todos los permisos de lista de control de acceso (ACL) de Kafka necesarios que se indican a continuación:

- Clúster DescribeConfigs
- Descripción del grupo
- Grupo de lectura
- Descripción del tema
- Tema de lectura

Para el control de acceso de IAM, faltan uno o más de los permisos necesarios en el rol de ejecución de la función para acceder al grupo o al tema. Consulte la lista de permisos necesarios en [the section called “Autenticación basada en roles de IAM”](#).

Cuando crea las ACL de Kafka o una política de IAM con los permisos de clúster de Kafka necesarios, debe especificar el tema y el grupo como recursos. El nombre del tema debe coincidir con el tema en la asignación de origen de eventos. El nombre del grupo debe coincidir con el UUID de la asignación de origen de eventos.

Después de agregar los permisos necesarios al rol de ejecución, pueden pasar varios minutos hasta que los cambios surtan efecto.

## Error de autenticación de SASL

Para SASL/SCRAM, este error indica que el nombre de usuario y la contraseña proporcionados no son válidos.

Para el control de acceso de IAM, falta el permiso `kafka-cluster:Connect` para el clúster de MSK en el rol de ejecución. Agregue este permiso al rol y especifique el nombre de recurso de Amazon (ARN) del clúster como recurso.

Es posible que vea que este error se produce de forma intermitente. El clúster rechaza las conexiones después de que el número de conexiones TCP supere la [cuota de servicio de Amazon MSK](#). Lambda retrocede y vuelve a intentarlo hasta que una conexión tenga éxito. Después de que Lambda se conecte al clúster y sondee los registros, el último resultado de procesamiento cambia a OK.

### El servidor no pudo autenticar Lambda

Este error indica que los agentes de Kafka de Amazon MSK no se han podido autenticar con Lambda. Este error puede producirse por cualquiera de las razones siguientes:

- No proporcionó ningún certificado de cliente para la autenticación de mTLS.
- Proporcionó un certificado de cliente, pero los agentes no están configurados para utilizar mTLS.
- Los agentes no confían en el certificado de cliente.

### La clave privada o el certificado proporcionados no son válidos

Este error indica que el consumidor de Amazon MSK no ha podido utilizar el certificado ni la clave privada proporcionados. Asegúrese de que el formato del certificado y la clave sea PEM y de que el cifrado de clave privada utilice un algoritmo PBES1.

## Configuración de red

Para que Lambda utilice su clúster de Kafka como origen de eventos, necesita acceder a la Amazon VPC en la que reside su clúster. Le recomendamos que implemente [puntos de conexión de VPC](#) AWS PrivateLink para que Lambda acceda a su VPC. Implemente puntos de conexión para Lambda y AWS Security Token Service (AWS STS). Si el agente utiliza autenticación, implemente también un punto de conexión de VPC para Secrets Manager. Si configuró un [destino en caso de error](#), implemente también un punto de conexión de VPC para el servicio de destino.

De manera alternativa, asegúrese de que la VPC asociada a su clúster de Kafka incluya una gateway NAT por subred pública. Para obtener más información, consulte [the section called “Acceso a Internet para funciones de VPC”](#).

Si usa puntos de enlace de VPC, también debe configurarlos para [habilitar los nombres DNS privados](#).

Al crear una asignación de orígenes de eventos para un clúster de MSK, Lambda comprueba si las interfaces de red elásticas (ENI) ya están presentes en las subredes y los grupos de seguridad de la VPC de su clúster. Si Lambda encuentra ENI existentes, intenta reutilizarlos. De lo contrario, Lambda crea nuevos ENI para conectarse al origen de eventos e invocar la función.

#### Note

Las funciones de Lambda siempre se ejecutan dentro de VPC propiedad del servicio de Lambda. El servicio mantiene estas VPC de forma automática, y no son visibles para los clientes. También puede conectar su función a una Amazon VPC. En cualquier caso, la configuración de VPC de la función no afecta la asignación de orígenes de eventos. Solo la configuración de la VPC del origen de eventos determina cómo Lambda se conecta al origen de eventos.

La configuración de Amazon VPC se puede detectar a través de la [API de Amazon MSK](#). No tiene que configurarla durante la configuración con el comando `create-event-source-mapping`.

Para obtener más información sobre la configuración de la red, consulte [Configuración de AWS Lambda con un clúster Apache Kafka dentro de una VPC](#) en el Blog de informática de AWS.

## Reglas de los grupos de seguridad de VPC

Configure los grupos de seguridad de Amazon VPC que contienen su clúster con las siguientes reglas (como mínimo):

- Reglas de entrada: permiten todo el tráfico en el puerto del agente de Amazon MSK (9092 para texto sin formato, 9094 para TLS, 9096 para SASL, 9098 para IAM) para los grupos de seguridad especificados para el origen de eventos.
- Reglas de salida: permiten todo el tráfico en el puerto 443 para todos los destinos. Permiten todo el tráfico en el puerto del agente de Amazon MSK (9092 para texto sin formato, 9094 para TLS, 9096 para SASL, 9098 para IAM) para los grupos de seguridad especificados para el origen de eventos.



- Si utiliza puntos de conexión de VPC en lugar de una puerta de enlace NAT, los grupos de seguridad asociados a los puntos de conexión de VPC deben permitir todo el tráfico entrante en el puerto 443 desde los grupos de seguridad del origen de eventos.

## Uso de los puntos de enlace de la VPC

Cuando utiliza puntos de conexión de VPC, las llamadas a la API para invocar su función se enrutan a través de estos puntos de conexión mediante los ENI. La entidad principal del servicio de Lambda debe llamar a `sts:AssumeRole` y `lambda:InvokeFunction` para cualquier rol y función que utilicen esos ENI.

De forma predeterminada, los puntos de conexión de VPC tienen políticas de IAM abiertas. La práctica recomendada es restringir estas políticas para permitir que solo entidades principales específicas realicen las acciones necesarias mediante ese punto de conexión. Para garantizar que la asignación de orígenes de eventos pueda invocar la función de Lambda, la política de puntos de conexión de VPC debe permitir que el principio del servicio de Lambda llame a `sts:AssumeRole` y `lambda:InvokeFunction`. Restringir las políticas de puntos de conexión de VPC para permitir únicamente las llamadas a la API que se originen en su organización impide que la asignación de orígenes de eventos funcione correctamente.

En el siguiente ejemplo de políticas de puntos de conexión de VPC, se muestra cómo conceder el acceso necesario a las entidades principales del servicio de Lambda para AWS STS y los puntos de conexión de Lambda.

### Example Política del punto de conexión de VPC: punto de conexión de AWS STS

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

## Example Política del punto de conexión de VPC: punto de conexión de Lambda

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

Si su agente de Kafka utiliza la autenticación, también puede restringir la política de puntos de conexión de VPC para el punto de conexión de Secrets Manager. Para llamar a la API de Secrets Manager, Lambda usa su rol de función, no la entidad principal de servicio de Lambda. En el siguiente ejemplo, se muestra una política del punto de conexión de Secrets Manager.

## Example Política de punto de conexión de VPC: punto de conexión de Secrets Manager

```
{
  "Statement": [
    {
      "Action": "secretsmanager:GetSecretValue",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "customer_function_execution_role_arn"
        ]
      },
      "Resource": "customer_secret_arn"
    }
  ]
}
```

Si tiene configurado un destino en caso de error, Lambda también utiliza el rol de su función para llamar a `s3:PutObject`, `sns:Publish` o `sqs:sendMessage` mediante los ENI administrados por Lambda.

# Procesamiento de mensajes de Amazon MSK con Lambda

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

## Temas

- [Agregar Amazon MSK como origen de eventos](#)
- [Parámetros de configuración de Amazon MSK](#)
- [Creación de asignaciones de orígenes de eventos entre cuentas](#)
- [Utilizar un clúster de Amazon MSK como origen de eventos](#)
- [Posiciones iniciales de flujos y sondeo](#)
- [Métricas de Amazon CloudWatch](#)
- [Escalado automático del origen de eventos de Amazon MSK](#)

## Agregar Amazon MSK como origen de eventos

Para crear una [asignación de orígenes de eventos](#), agregue Amazon MSK como un [desencadenador](#) de la función de Lambda a través de la consola de Lambda, un [AWS SDK](#), o el [AWS Command Line Interface \(AWS CLI\)](#). Tenga en cuenta que cuando agrega Amazon MSK como desencadenador, Lambda asume la configuración de VPC del clúster de Amazon MSK, no la configuración de VPC de la función de Lambda.

En esta sección se describe cómo crear una asignación de orígenes de eventos mediante la consola de Lambda y la AWS CLI.

## Requisitos previos

- Un clúster Amazon MSK y un tema Kafka. Para obtener más información, consulte [Introducción al uso de Amazon MSK](#) en la Guía para desarrolladores de Amazon Managed Streaming for Apache Kafka.
- Un [rol de ejecución](#) con permiso para acceder a los recursos de AWS que utiliza el clúster de MSK.

## ID del grupo de consumidores personalizable

Al configurar Kafka como origen de eventos, puede especificar un ID de grupo de consumidores. Este ID de grupo de consumidores es un identificador existente para el grupo de consumidores de Kafka al que desea que se una la función de Lambda. Puede utilizar esta característica para migrar sin problemas cualquier configuración de procesamiento de registro de Kafka en curso de otros consumidores a Lambda.

Si especifica un ID de grupo de consumidores y hay otros sondeadores activos dentro de ese grupo de consumidores, Kafka distribuirá los mensajes entre todos los consumidores. En otras palabras, Lambda no recibe todos los mensajes del tema Kafka. Si desea que Lambda gestione todos los mensajes del tema, desactive los demás sondeadores de ese grupo de consumidores.

Además, si especifica un ID de grupo de consumidores y Kafka encuentra un grupo de consumidores existente válido con el mismo ID, Lambda ignora el parámetro `StartingPosition` para la asignación de orígenes de eventos. En cambio, Lambda comienza a procesar los registros de acuerdo con la compensación comprometida del grupo de consumidores. Si especifica un ID de grupo de consumidores y Kafka no puede encontrar un grupo de consumidores existente, Lambda configura el origen de eventos con el `StartingPosition` especificado.

El ID del grupo de consumidores que especifique debe ser único entre todos los orígenes de eventos de Kafka. Tras crear una asignación de origen de eventos de Kafka con el ID de grupo de consumidores especificado, no puede actualizar este valor.

### Agregar un desencadenador de Amazon MSK (consola)

Siga estos pasos para agregar su clúster de Amazon MSK y un tema Kafka como desencadenador de su función de la función de Lambda.

### Cómo agregar un desencadenador Amazon MSK a su función de Lambda (consola)

1. Abra la página de [Functions \(Funciones\)](#) en la consola de Lambda.
2. Elija el nombre de su función de Lambda.
3. En Descripción general de la función, elija Agregar desencadenador.
4. En Configuración del desencadenador, haga lo siguiente:
  - a. Elija el tipo de desencadenador MSK.
  - b. Para clúster de MSK, seleccione su clúster.

- c. Para el Tamaño del lote, establezca el número máximo de mensajes para recibir un solo lote.
  - d. Para el periodo de lotes, ingrese la cantidad máxima de segundos que Lambda emplea a fin de recopilar registros antes de invocar la función.
  - e. Para Nombre de tema, escriba un nombre para el tema Kafka.
  - f. (Opcional) Para el ID del grupo de consumidores, ingrese el ID de un grupo de consumidores de Kafka al que unirse.
  - g. (Opcional) En Posición inicial, elija Última para empezar a leer el flujo desde el registro más reciente, Horizonte de supresión para comenzar por el registro más antiguo disponible o En la marca de tiempo para especificar una marca de tiempo desde la cual comenzar a leer.
  - h. (Opcional) En Authentication (Autenticación), elija la clave secreta para la autenticación con los agentes en el clúster de MSK.
  - i. Para crear el desencadenador en un estado deshabilitado para la prueba (recomendado), desactive Activar desencadenador. O bien, para habilitar el desencadenador de inmediato, seleccione Activar desencadenador.
5. Para crear el desencadenador, elija Add (Añadir).

### Agregar un desencadenador de Amazon MSK (AWS CLI)

Utilice los siguientes comandos AWS CLI de ejemplo para crear y ver un desencadenador Amazon MSK para su función de a función Lambda.

#### Crear un desencadenador mediante el AWS CLI

Example — Creación de una asignación de orígenes de eventos para un clúster que utilice la autenticación de IAM

El siguiente ejemplo utiliza el comando [create-event-source-mapping](#) AWS CLI para mapear una función de Lambda llamada `my-kafka-function` a un tema de Kafka llamado `AWSKafkaTopic`. La posición inicial del tema está establecida en `LATEST`. Cuando el clúster utiliza la [autenticación basada en roles de IAM](#), no se necesita un objeto [SourceAccessConfiguration](#).

Ejemplo:

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --
```

```
--starting-position LATEST \  
--function-name my-kafka-function
```

Example — Creación de una asignación de orígenes de eventos para un clúster que utilice la autenticación SASL/SCRAM

Si el clúster usa la [autenticación SASL/SCRAM](#), debe incluir un objeto [SourceAccessConfiguration](#) que especifique SASL\_SCRAM\_512\_AUTH y un ARN secreto de Secrets Manager.

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  
  --source-access-configurations '["Type": "SASL_SCRAM_512_AUTH", "URI":  
"arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"]]'
```

Example — Creación de una asignación de orígenes de eventos para un clúster que utilice la autenticación mTLS

Si el clúster usa la [autenticación mTLS](#), debe incluir un objeto [SourceAccessConfiguration](#) que especifique CLIENT\_CERTIFICATE\_TLS\_AUTH y un ARN secreto de Secrets Manager.

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-  
fd1b-45ad-85dd-15b4a5a6247e-2 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  
  --source-access-configurations '["Type": "CLIENT_CERTIFICATE_TLS_AUTH", "URI":  
"arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"]]'
```

Para obtener más información, consulte la documentación de referencia de la API [CreateEventSourceMapping](#).

Visualización del estado mediante la AWS CLI

El siguiente ejemplo utiliza el comando [get-event-source-mapping](#) AWS CLI para describir el estado de la asignación de orígenes de eventos que ha creado.

```
aws lambda get-event-source-mapping \
  --uuid 6d9bce8e-836b-442c-8070-74e77903c815
```

## Parámetros de configuración de Amazon MSK

Todos los tipos de origen de eventos Lambda comparten las mismas operaciones

[CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Amazon MSK.

Parámetro	Obligatoria	Predeterminado	Notas
AmazonManagedKafkaEventSourceConfig	N	Contiene el campo ConsumerGroupId, que se establece de forma predeterminada en un valor único.	Solo se puede establecer en Crear
BatchSize	N	100	Máximo: 10 000
Habilitado	N	Habilitado	Ninguno
EventSourceArn	Y	N/A	Solo se puede establecer en Crear
FunctionName	Y	N/A	Ninguno
FilterCriteria	N	N/A	<a href="#">Controle qué eventos envía Lambda a la función</a>
MaximumBatchingWindowInSeconds	N	500 ms	<a href="#">Comportamiento de procesamiento por lotes</a>
SourceAccessConfigurations	N	Sin credenciales	Credenciales de autenticación de SASL/SCRAM o CLIENT_CERTIFICATE

Parámetro	Obligatoria	Predeterminado	Notas
			_TLS_AUTH (MutualTLS) para el origen de eventos
StartingPosition	Y	N/A	AT_TIMESTAMP, TRIM_HORIZON o LATEST  Solo se puede establecer en Crear
StartingPositionTimestamp	N	N/A	Obligatorio si StartingPosition se establece en AT_TIMESTAMP
Temas	Y	N/A	Nombre del tema de Kafka  Solo se puede establecer en Crear

## Creación de asignaciones de orígenes de eventos entre cuentas

Puede utilizar la [conectividad privada de varias VPC](#) para conectar una función de Lambda a un clúster de MSK provisionado en otra Cuenta de AWS. La conectividad de varias VPC utiliza AWS PrivateLink, que mantiene todo el tráfico dentro de la red de AWS.

### Note

No puede crear asignaciones de orígenes de eventos entre cuentas para clústeres de MSK sin servidor.

Para crear una asignación de orígenes de eventos entre cuentas, primero debe [configurar la conectividad de múltiples VPC para el clúster de MSK](#). Al crear la asignación de orígenes de eventos, utilice el ARN de conexión de VPC administrada en lugar del ARN del clúster, como se muestra en



los siguientes ejemplos. La operación [CreateEventSourceMapping](#) también varía según el tipo de autenticación que utilice el clúster de MSK.

Example — Creación de una asignación de orígenes de eventos entre cuentas para un clúster que utilice la autenticación de IAM

Cuando el clúster utiliza la [autenticación basada en roles de IAM](#), no se necesita un objeto [SourceAccessConfiguration](#). Ejemplo:

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
  my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
```

Example — Creación de una asignación de orígenes de eventos entre cuentas para un clúster que utilice la autenticación SASL/SCRAM

Si el clúster usa la [autenticación SASL/SCRAM](#), debe incluir un objeto [SourceAccessConfiguration](#) que especifique SASL\_SCRAM\_512\_AUTH y un ARN secreto de Secrets Manager.

Hay dos formas de utilizar los secretos para las asignaciones de orígenes de eventos entre cuentas de Amazon MSK con autenticación SASL/SCRAM:

- Cree un secreto en la cuenta de la función de Lambda y sincronícelo con el secreto del clúster. [Cree una rotación](#) para mantener los dos secretos sincronizados. Esta opción le permite controlar el secreto desde la cuenta de la función.
- Use el secreto asociado al clúster de MSK. Este secreto debe permitir el acceso entre cuentas a la cuenta de la función de Lambda. Para obtener más información, consulte [Permisos para secretos de AWS Secrets Manager para usuarios en una cuenta diferente](#).

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
  my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH", "URI":
  "arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

Example — Creación de una asignación de orígenes de eventos entre cuentas para un clúster que utilice la autenticación mTLS

Si el clúster usa la [autenticación mTLS](#), debe incluir un objeto [SourceAccessConfiguration](#) que especifique `CLIENT_CERTIFICATE_TLS_AUTH` y un ARN secreto de Secrets Manager. El secreto se puede almacenar en la cuenta del clúster o en la cuenta de la función de Lambda.

```
aws lambda create-event-source-mapping \  
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/  
  my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \  
  --topics AWSKafkaTopic \  
  --starting-position LATEST \  
  --function-name my-kafka-function \  
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH", "URI":  
  "arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

## Utilizar un clúster de Amazon MSK como origen de eventos

Cuando agrega su clúster de Apache Kafka o Amazon MSK como desencadenador para su función de Lambda, el clúster se utiliza como [origen de eventos](#).

Lambda lee los datos de eventos de los temas de Kafka que especifique como `Topics` en una solicitud de [CreateEventSourceMapping](#), en función de la `StartingPosition` que especifique. Después de un procesamiento exitoso, su tema de Kafka se compromete a su clúster de Kafka.

Si especifica la `StartingPosition` como `LATEST`, Lambda comienza a leer el último mensaje de cada partición que pertenece al tema. Debido a que puede haber algún retraso después de la configuración del desencadenador antes de que Lambda comience a leer los mensajes, Lambda no lee ningún mensaje producido durante este plazo.

Lambda lee los mensajes secuencialmente para cada partición de tema de Kafka. Una sola carga de Lambda puede contener mensajes de varias particiones. Cuando hay más registros disponibles, Lambda continúa procesando registros en lotes, en función del valor `BatchSize` que especifique en una solicitud de [CreateEventSourceMapping](#), hasta que la función se ponga al día con el tema.

Después de que Lambda procese cada lote, confirma los desplazamientos de los mensajes en ese lote. Si su función devuelve un error para cualquiera de los mensajes de un lote, Lambda reintenta todo el lote de mensajes hasta que el procesamiento sea correcto o los mensajes caduquen. Puede enviar los registros con error en todos los reintentos a un [destino en caso de error](#) para su posterior procesamiento.

**Note**

Si bien las funciones de Lambda suelen tener un límite de tiempo de espera máximo de 15 minutos, las asignaciones de orígenes de eventos para Amazon MSK, Apache Kafka autoadministrado, Amazon DocumentDB y Amazon MQ para ActiveMQ y RabbitMQ solo admiten funciones con límites de tiempo de espera máximos de 14 minutos. Esta restricción garantiza que la asignación de orígenes de eventos pueda gestionar correctamente los errores y reintentos de las funciones.

## Posiciones iniciales de flujos y sondeo

Tenga en cuenta que el sondeo de flujos durante la creación y las actualizaciones de la asignación de orígenes de eventos es, en última instancia, coherente.

- Durante la creación de la asignación de orígenes de eventos, es posible que se demore varios minutos en iniciar el sondeo de los eventos del flujo.
- Durante las actualizaciones de la asignación de orígenes de eventos, es posible que se demore varios minutos en detener y reiniciar el sondeo de los eventos del flujo.

Este comportamiento significa que, si especifica LATEST como posición inicial del flujo, la asignación de orígenes de eventos podría omitir eventos durante la creación o las actualizaciones. Para garantizar que no se pierda ningún evento, especifique la posición inicial del flujo como TRIM\_HORIZON o AT\_TIMESTAMP.

## Métricas de Amazon CloudWatch

Lambda emite la métrica `OffsetLag` mientras la función procesa los registros. El valor de esta métrica es la diferencia de compensación entre el último registro escrito en el tema de origen de eventos de Kafka y el último registro que procesó el grupo de consumidores de su función. Puede utilizar `OffsetLag` para estimar la latencia entre el momento en el que se agrega un registro y el momento en el que el grupo lo procesa.

Una tendencia ascendente en `OffsetLag` puede indicar problemas con los sondeadores en el grupo de consumidores de su función. Para obtener más información, consulte [Ver las métricas de funciones de Lambda](#).

## Escalado automático del origen de eventos de Amazon MSK

Al crear inicialmente un origen de eventos de Amazon MSK, Lambda asigna un consumidor para procesar todas las particiones del tema de Kafka. Cada consumidor tiene varios procesadores que se ejecutan en paralelo para gestionar el aumento de las cargas de trabajo. Además, Lambda escala o reduce verticalmente de manera automática el número de consumidores, en función de la carga de trabajo. Para conservar el orden de mensajes en cada partición, el número máximo de consumidores es un consumidor por partición en el tema.

En intervalos de un minuto, Lambda evalúa el retraso de compensación del consumidor de todas las particiones del tema. Si el retraso es demasiado alto, la partición recibe mensajes más rápido de lo que Lambda puede procesarlos. Si es necesario, Lambda agrega o elimina a los consumidores del tema. El proceso de escalado para agregar o eliminar consumidores se produce dentro de los tres minutos posteriores a la evaluación.

Si su función de Lambda objetivo está limitada, Lambda reduce el número de consumidores. Esta acción reduce la carga de trabajo de la función al reducir el número de mensajes que los consumidores pueden recuperar y enviar a la función.

Para monitorear el rendimiento de su tema de Kafka, consulte la [métrica de retraso de desplazamiento](#) que Lambda emite mientras la función procesa los registros.

Para comprobar cuántas invocaciones de función se producen en paralelo, también puede supervisar las [métricas de simultaneidad](#) para su función.

## Filtrado de eventos con un origen de eventos de Amazon MSK

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo se filtran los eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para los orígenes de eventos de Amazon MSK.

### Temas

- [Conceptos básicos del filtrado de eventos de Amazon MSK](#)

## Conceptos básicos del filtrado de eventos de Amazon MSK

Supongamos que un productor escribe mensajes a un tema de su clúster de Amazon MSK, ya sea en formato JSON válido o como cadenas simples. Un registro de ejemplo tendría el siguiente aspecto, con el mensaje convertido en una cadena codificada en Base64 en el campo `value`.

```
{
  "mytopic-0": [
    {
      "topic": "mytopic",
      "partition": 0,
      "offset": 15,
      "timestamp": 1545084650987,
      "timestampType": "CREATE_TIME",
      "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
      "headers": []
    }
  ]
}
```

Supongamos que su productor de Apache Kafka escribe mensajes a su tema en el siguiente formato JSON.

```
{
  "device_ID": "AB1234",
  "session": {
    "start_time": "yyyy-mm-ddThh:mm:ss",
    "duration": 162
  }
}
```

Puede utilizar la clave `value` para filtrar registros. Supongamos que desea filtrar solo los registros en los que `device_ID` comience con las letras AB. El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "value": {
    "device_ID": [ { "prefix": "AB" } ]
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

## Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/  
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :  
[ { \"prefix\": \"AB\" } ] } }"]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :  
[ { \"prefix\": \"AB\" } ] } }"]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix": "AB" } ] } }'
```

Con Amazon MSK, también puede filtrar los registros en los que el mensaje sea una cadena simple. Supongamos que desea ignorar los mensajes en los que la cadena es "error". El objeto `FilterCriteria` tendría el siguiente aspecto.

```
{
  "Filters": [
    {
      "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "value": [
    {
      "anything-but": [ "error" ]
    }
  ]
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \  
  --function-name my-function \  
  --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/  
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":  
[ \"error\" ] } ] }"]}'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \  
  --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":  
[ \"error\" ] } ] }"]}'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:  
  Filters:  
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

Los mensajes de Amazon MSK deben ser cadenas codificadas en UTF-8, cadenas simples o en formato JSON. Esto se debe a que Lambda decodifica las matrices de bytes de Amazon MSK en UTF-8 antes de aplicar los criterios de filtro. Si los mensajes utilizan otra codificación, como UTF-16 o ASCII, o el formato del mensaje no coincide con el formato de `FilterCriteria`, Lambda solo procesa los filtros de metadatos. En la siguiente tabla se resume el comportamiento específico:



Formato del mensaje entrante	Formato del patrón de filtro para las propiedades del mensaje	Acción resultante
Cadena sin formato	Cadena sin formato	Lambda filtra en función de los criterios de filtro.
Cadena sin formato	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
Cadena sin formato	JSON válido	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Cadena sin formato	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.
Cadena no codificada con UTF-8	JSON, cadena sin formato o sin patrón	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.

## Capturar lotes descartados para un origen de eventos de Amazon MSK

Para retener los registros de las invocaciones de asignación de orígenes de eventos fallidos, agregue un destino a la asignación de orígenes de eventos de su función. Cada registro enviado al destino es un documento JSON con metadatos sobre la invocación fallida. Puede configurar cualquier tema de Amazon SNS, cola de Amazon SQS o bucket de S3 como destino. Su rol de ejecución debe tener permisos para el destino:

- Para destinos de SQS: [sqs:SendMessage](#)
- Para destinos de SNS: [sns:Publish](#)
- Para destinos de bucket de S3: [s3:PutObject](#) y [s3:ListBuckets](#)

Además, si configuró una clave de KMS en su destino, Lambda necesita los siguientes permisos según el tipo de destino:

- Si tiene activado el cifrado con su propia clave de KMS para un destino de S3, necesitará [kms:GenerateDataKey](#). Si la clave de KMS y el destino del bucket de S3 están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:GenerateDataKey`.
- Si tiene activado el cifrado con su propia clave de KMS para un destino de SQS, necesitará [kms:Decrypt](#) y [kms:GenerateDataKey](#). Si la clave de KMS y el destino de la cola de SQS están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:Decrypt`, `kms:GenerateDataKey`, [kms:DescribeKey](#) y [kms:ReEncrypt](#).
- Si tiene activado el cifrado con su propia clave de KMS para un destino de SNS, necesitará [kms:Decrypt](#) y [kms:GenerateDataKey](#). Si la clave de KMS y el destino del tema de SNS están en una cuenta diferente a la de su función de Lambda y rol de ejecución, configure la clave de KMS para que confíe en el rol de ejecución y permita `kms:Decrypt`, `kms:GenerateDataKey`, [kms:DescribeKey](#) y [kms:ReEncrypt](#).

## Configuración de destinos en caso de error para la asignación de orígenes de eventos de Amazon MSK

Para configurar un destino en caso de error mediante la consola, siga estos pasos:

1. Abra la [página de Funciones](#) en la consola de Lambda.

2. Elija una función.
3. En Descripción general de la función, elija Agregar destino.
4. En Origen, elija Invocación de asignación de orígenes de eventos.
5. Para la Asignación de orígenes de eventos, elija un origen de eventos que esté configurado para esta función.
6. En Condición, seleccione En caso de error. Para las invocaciones de asignación de orígenes de eventos, esta es la única condición aceptada.
7. En Tipo de destino, elija el tipo de destino al que Lambda envía los registros de invocación.
8. En Destino, elija un recurso.
9. Seleccione Guardar.

También puede configurar un destino en caso de error mediante la AWS CLI. Por ejemplo, el siguiente comando [create-event-source-mapping](#) agrega una asignación de orígenes de eventos con un destino de SQS en caso de error a MyFunction:

```
aws lambda create-event-source-mapping \  
--function-name "MyFunction" \  
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/  
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-  
east-1:123456789012:dest-queue"}}'
```

El siguiente comando [UpdateEventSourceMapping](#) agrega un destino S3 en caso de error al origen de eventos asociado a la entrada uuid:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Para eliminar un destino, introduzca una cadena vacía como argumento del parámetro `destination-config`:

```
aws lambda update-event-source-mapping \  
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \  
--destination-config '{"OnFailure": {"Destination": ""}}'
```

## Ejemplo de registro de invocación SNS y SQS

El siguiente ejemplo muestra lo que Lambda envía a un destino de tema de SNS o cola de SQS cuando se produce un error en la invocación de un origen de eventos de Kafka. Cada una de las claves en `recordsInfo` contiene el tema y la partición de Kafka, separados por un guion. Por ejemplo, para la clave `"Topic-0"`, `Topic` es el tema de Kafka, y `0` es la partición. Para cada tema y partición, puede usar los desplazamientos y los datos de las marcas de tiempo para buscar los registros de invocación originales.

```
{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted | MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
      },
      "Topic-1": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",

```

```

        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
    }
}
}
}

```

### Ejemplo de registro de invocación de un destino S3

Para los destinos de S3, Lambda envía todo el registro de invocación junto con los metadatos al destino. El siguiente ejemplo muestra lo que Lambda envía a un bucket de S3 de destino cuando se produce un error en la invocación de un origen de evento de Kafka. Además de todos los campos del ejemplo anterior para los destinos de SQS y SNS, el campo `payload` contiene el registro de invocación original en forma de cadena JSON de escape.

```

{
  "requestContext": {
    "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
    "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
    "approximateInvokeCount": 1
  },
  "responseContext": { // null if record is MaximumPayloadSizeExceeded
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T00:38:06.021Z",
  "KafkaBatchInfo": {
    "batchSize": 500,
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
    "bootstrapServers": "...",
    "payloadSize": 2039086, // In bytes
    "recordsInfo": {
      "Topic-0": {
        "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
        "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
        "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
        "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",

```

```
    },
    "Topic-1": {
      "firstRecordOffset":
"49601189658422359378836298521827638475320189012309704722",
      "lastRecordOffset":
"49601189658422359378836298522902373528957594348623495186",
      "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
      "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
    }
  }
},
"payload": "<Whole Event>" // Only available in S3
}
```

### Tip

También se recomienda habilitar el control de versiones de S3 en el bucket de destino.

## Tutorial: Uso de una asignación de orígenes de eventos de Amazon MSK para invocar una función de Lambda

En este tutorial, hará lo siguiente:

- Creará una función de Lambda en la misma cuenta de AWS de un clúster existente de Amazon MSK.
- Configuraré las redes y la autenticación para que Lambda se comuniquen con Amazon MSK.
- Configuraré una asignación de orígenes de eventos de Amazon MSK de Lambda que ejecute la función de Lambda cuando los eventos aparezcan en el tema.

Una vez finalizados estos pasos, cuando los eventos se envíen a Amazon MSK, podrá configurar una función de Lambda para procesar dichos eventos automáticamente con su propio código de Lambda personalizado.

¿Qué puede hacer con esta característica?

Ejemplo de solución: utilice una asignación de orígenes de eventos de MSK para ofrecer puntuaciones en directo a sus clientes.

Considere el siguiente escenario: su empresa aloja una aplicación web en la que sus clientes pueden ver información sobre eventos en directo, como partidos deportivos. Las actualizaciones de información del juego se proporcionan a su equipo a través de un tema de Kafka en Amazon MSK. Desea diseñar una solución que utilice las actualizaciones del tema de MSK para ofrecer a los clientes una visión actualizada del evento en directo desde una aplicación que desarrolle. Ha optado por el siguiente enfoque de diseño: sus aplicaciones cliente se comunicarán con un backend sin servidor alojado en AWS. Los clientes se conectarán a través de sesiones de websocket mediante la API de WebSocket de Amazon API Gateway.

En esta solución, necesita un componente que lea los eventos de MSK, ejecute una lógica personalizada para preparar esos eventos para la capa de aplicación y, a continuación, reenvíe esa información a la API de API Gateway. Puede implementar este componente con AWS Lambda al proporcionar su lógica personalizada en una función de Lambda y luego llamarla con una asignación de orígenes de eventos de Amazon MSK de AWS Lambda.

Para obtener más información sobre la implementación de soluciones mediante la API de WebSocket de Amazon API Gateway, consulte los [tutoriales de la API de WebSocket](#) en la documentación de API Gateway.

## Requisitos previos

Una cuenta de AWS con los siguientes recursos preconfigurados:

Para cumplir estos requisitos previos, le recomendamos que consulte la sección [Getting started using Amazon MSK](#) en la documentación de Amazon MSK.

- Un clúster de Amazon MSK Consulte [Create an Amazon MSK cluster](#) en Getting started using Amazon MSK.
- La siguiente configuración:
  - Asegúrese de que la autenticación basada en roles de IAM esté habilitada en la configuración de seguridad de su clúster. Esto mejora la seguridad al limitar la función de Lambda para que solo acceda a los recursos de Amazon MSK necesarios. Esto está habilitado de forma predeterminada en los nuevos clústeres de Amazon MSK.
  - Asegúrese de que el acceso público esté desactivado en la configuración de red del clúster. Restringir el acceso de su clúster de Amazon MSK a Internet mejora su seguridad al limitar el número de intermediarios que gestionan sus datos. Esto está habilitado de forma predeterminada en los nuevos clústeres de Amazon MSK.

- Un tema de Kafka en su clúster de Amazon MSK para usarlo en esta solución. Consulte [Create a topic](#) en Getting started using Amazon MSK.
- Un host de administración de Kafka configurado para recuperar información de su clúster de Kafka y enviar los eventos de Kafka a su tema para probarlos, como una instancia de Amazon EC2 con la CLI de administración de Kafka y la biblioteca de IAM de Amazon MSK instaladas. Consulte [Create an Amazon MSK cluster](#) en Getting started using Amazon MSK.

Una vez que haya configurado estos recursos, recopile la siguiente información de su cuenta de AWS para confirmar que puede continuar.

- El nombre de su clúster de Amazon MSK. Puede encontrar esta información en la consola de Amazon MSK.
- El UUID del clúster, que forma parte del ARN de su clúster de Amazon MSK, que puede encontrar en la consola de Amazon MSK. Siga los procedimientos que se indican en [Listing clusters](#) en la documentación de Amazon MSK para encontrar esta información.
- Los grupos de seguridad asociados con su clúster de Amazon MSK. Puede encontrar esta información en la consola de Amazon MSK. En los siguientes pasos, se hará referencia a ellos como *clusterSecurityGroups*.
- El identificador de la Amazon VPC que contiene su clúster de Amazon MSK. Para encontrar esta información, identifique las subredes asociadas a su clúster de Amazon MSK en la consola de Amazon MSK y, a continuación, identifique la Amazon VPC asociada a la subred en la consola de Amazon VPC.
- El nombre del tema de Kafka utilizado en su solución. Puede encontrar esta información al llamar a su clúster de Amazon MSK con la CLI de `topics` de Kafka desde su host de administración de Kafka. Para obtener más información sobre la CLI de temas, consulte [Adding and removing topics](#) en la documentación de Kafka.
- El nombre de un grupo de consumidores para su tema de Kafka, adecuado para que lo utilice su función de Lambda. Lambda puede crear este grupo automáticamente, por lo que no es necesario crearlo con la CLI de Kafka. Si necesita administrar sus grupos de consumidores, para obtener más información sobre la CLI de grupos de consumidores, consulte [Managing Consumer Groups](#) en la documentación de Kafka.

Los siguientes permisos en su cuenta de AWS:

- Permiso para crear y administrar una función de Lambda

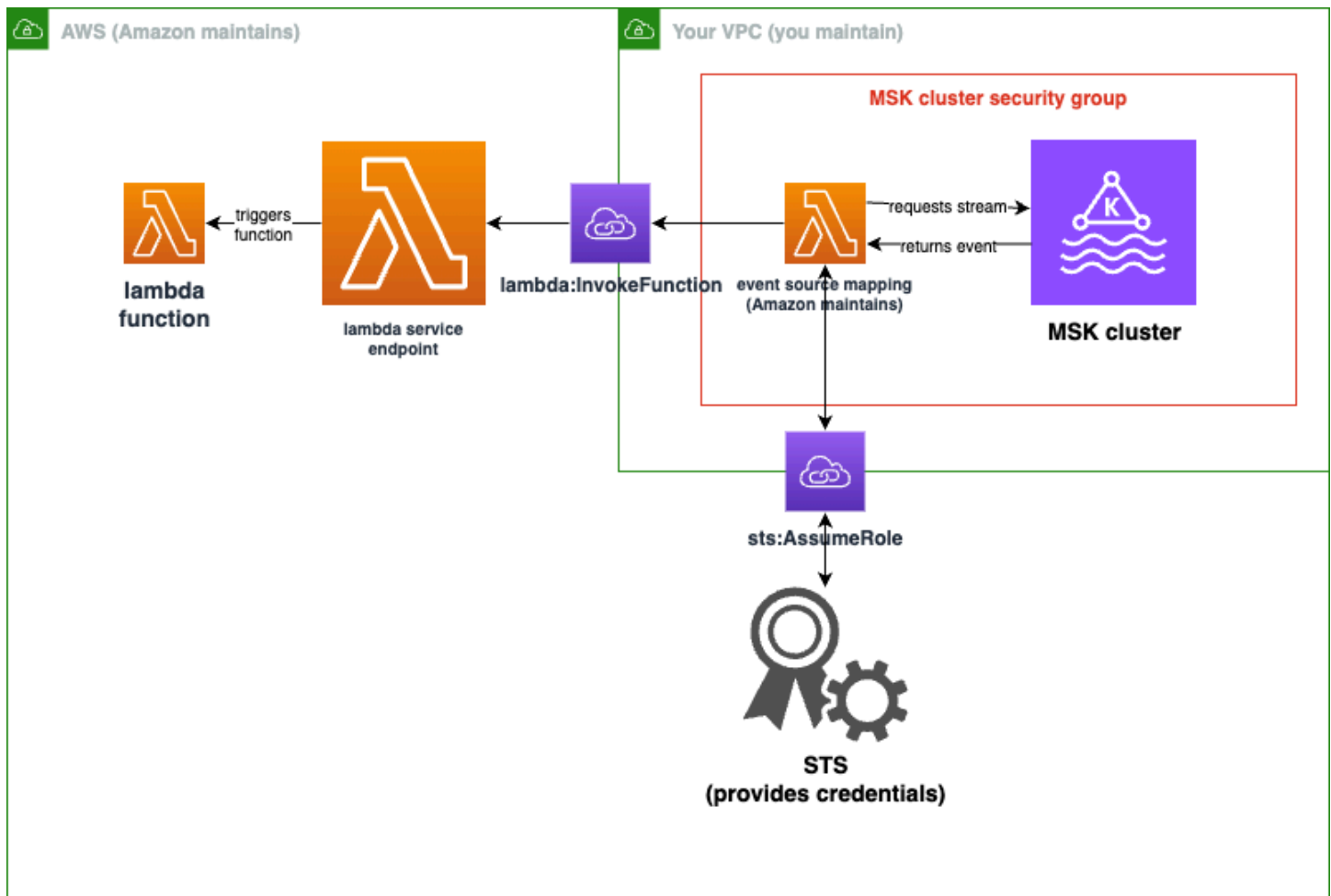


- Permiso para crear políticas de IAM y asociarlas a su función de Lambda.
- Permiso para crear puntos de conexión de Amazon VPC y modificar la configuración de red en la Amazon VPC que aloja el clúster de Amazon MSK.

## Configuración de la conectividad de red para que Lambda se comuniquen con Amazon MSK

Use AWS PrivateLink para conectar Lambda y Amazon MSK. Para ello, puede crear puntos de conexión de Amazon VPC de interfaz en la consola de Amazon VPC. Para obtener más información acerca de la configuración de red, consulte [the section called “Configuración de red”](#).

Cuando se ejecuta una asignación de orígenes de eventos de Amazon MSK en nombre de una función de Lambda, asume el rol de ejecución de la función de Lambda. Este rol de IAM autoriza la asignación para acceder a los recursos protegidos por IAM, como su clúster de Amazon MSK. Si bien los componentes comparten un rol de ejecución, la asignación de Amazon MSK y la función de Lambda tienen requisitos de conectividad distintos para sus respectivas tareas, como se muestra en el siguiente diagrama.



La asignación de orígenes de eventos pertenece al grupo de seguridad de su clúster de Amazon MSK. En este paso de red, cree puntos de conexión de Amazon VPC a partir de la VPC de su clúster de Amazon MSK para conectar la asignación de orígenes de eventos a los servicios Lambda y STS. Proteja estos puntos de conexión para que acepten el tráfico del grupo de seguridad de su clúster de Amazon MSK. A continuación, ajuste los grupos de seguridad del clúster de Amazon MSK para permitir que la asignación de orígenes de eventos se comuniquen con el clúster de Amazon MSK.

Puede configurar los siguientes pasos que mediante la AWS Management Console.

Cómo configurar los puntos de conexión de Amazon VPC de interfaz para conectar Lambda y Amazon MSK

1. Cree un grupo de seguridad para los puntos de conexión de Amazon VPC de interfaz, *endpointSecurityGroup*, que permita el tráfico TCP entrante en 443 desde *clusterSecurityGroups*. Siga el procedimiento descrito en [Crear un grupo de seguridad](#) en la documentación de Amazon EC2 para crear un grupo de seguridad. A continuación, siga

el procedimiento descrito en [Agregar reglas a un grupo de seguridad](#) en la documentación de Amazon EC2 para agregar las reglas adecuadas.

Cree un grupo de seguridad con la siguiente información:

Al agregar las reglas de entrada, cree una regla para cada grupo de seguridad de *clusterSecurityGroups*. Para cada regla:

- En Tipo, seleccione HTTPS.
  - En Origen, seleccione uno de los *clusterSecurityGroups*.
2. Cree un punto de conexión que conecte el servicio de Lambda a la Amazon VPC que contiene su clúster de Amazon MSK. Siga el procedimiento que se indica en [Crear un punto de conexión de interfaz](#).

Cree un punto de conexión de interfaz con la siguiente información:

- En Nombre del servicio, seleccione `com.amazonaws.regionName.lambda`, cuyo *RegionName* sea donde se aloja la función de Lambda.
- Para VPC, seleccione la Amazon VPC que contiene el clúster de Amazon MSK.
- Para Grupos de seguridad, seleccione *endpointSecurityGroup*, el cual creó anteriormente.
- En Subredes, seleccione las subredes que alojan su clúster de Amazon MSK.
- En Política, proporcione el siguiente documento de política, que protege el punto de conexión para que lo utilice la entidad principal del servicio de Lambda para la acción `lambda:InvokeFunction`.

```
{
  "Statement": [
    {
      "Action": "lambda:InvokeFunction",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

```
}

```

- Asegúrese de que Habilitar nombre de DNS permanezca establecido.
3. Cree un punto de conexión que conecte el servicio AWS STS a la Amazon VPC que contiene su clúster de Amazon MSK. Siga el procedimiento que se indica en [Crear un punto de conexión de interfaz](#).

Cree un punto de conexión de interfaz con la siguiente información:

- En Nombre de servicio, seleccione AWS STS.
- Para VPC, seleccione la Amazon VPC que contiene el clúster de Amazon MSK.
- Para Grupos de seguridad, seleccione *endpointSecurityGroup*.
- En Subredes, seleccione las subredes que alojan su clúster de Amazon MSK.
- En Política, proporcione el siguiente documento de política, que protege el punto de conexión para que lo utilice la entidad principal del servicio de Lambda para la acción `sts:AssumeRole`.

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Resource": "*"
    }
  ]
}
```

- Asegúrese de que Habilitar nombre de DNS permanezca establecido.
4. Para cada grupo de seguridad asociado a su clúster de Amazon MSK, es decir, en *clusterSecurityGroups*, permita lo siguiente:
    - Permita que todo el tráfico TCP entrante y saliente de 9098 llegue a todos los *clusterSecurityGroups*, incluso dentro del propio grupo.
    - Permita todo el tráfico TCP saliente en 443.

Las reglas predeterminadas de los grupos de seguridad permiten parte de este tráfico, por lo que si el clúster está conectado a un único grupo de seguridad y ese grupo tiene reglas predeterminadas, no se necesitan reglas adicionales. Para ajustar las reglas de los grupos de seguridad, siga los procedimientos de [Agregar reglas a un grupo de seguridad](#) en la documentación de Amazon EC2.

Agregue reglas a sus grupos de seguridad con la siguiente información:

- Para cada regla de entrada o de salida del puerto 9098, proporcione lo siguiente:
  - En Type (Tipo), seleccione Custom TCP (TCP personalizada).
  - Para Intervalo de puertos, proporcione 9098.
  - En Origen, proporcione uno de los *clusterSecurityGroups*.
- Para cada regla de entrada del puerto 443, en Tipo, seleccione HTTPS.

## Creación de un rol de IAM para que Lambda lea un tema de Amazon MSK

Identifique los requisitos de autenticación que Lambda debe leer en su tema de Amazon MSK y, a continuación, defínalos en una política. Cree un rol, *lambdaAuthRole*, que autorice a Lambda a usar esos permisos. Autorice acciones en su clúster de Amazon MSK mediante las acciones de IAM `kafka-cluster`. A continuación, autorice a Lambda a realizar las acciones de Amazon MSK `kafka` y Amazon EC2 necesarias para descubrir el clúster de Amazon MSK y conectarse a él, así como las acciones de CloudWatch para que Lambda pueda registrar lo que ha hecho.

Cómo describir los requisitos de autenticación para que Lambda lea desde Amazon MSK

1. Redacte un documento de política de IAM (un documento JSON), *clusterAuthPolicy*, que permita a Lambda leer el tema de Kafka en su clúster de Amazon MSK mediante su grupo de consumidores de Kafka. Lambda requiere que se configure un grupo de consumidores de Kafka para la lectura.

Modifique la siguiente plantilla para adaptarla a sus requisitos previos:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

    "Action": [
        "kafka-cluster:Connect",
        "kafka-cluster:DescribeGroup",
        "kafka-cluster:AlterGroup",
        "kafka-cluster:DescribeTopic",
        "kafka-cluster:ReadData",
        "kafka-cluster:DescribeClusterDynamicConfiguration"
    ],
    "Resource": [
        "arn:aws:kafka:region:account-id:cluster/mskClusterName/cluster-uuid",
        "arn:aws:kafka:region:account-id:topic/mskClusterName/cluster-uuid/mskTopicName",
        "arn:aws:kafka:region:account-id:group/mskClusterName/cluster-uuid/mskGroupName"
    ]
}
]
}

```

Para obtener más información, consulte [the section called “Autenticación basada en roles de IAM”](#). Al redactar la política:

- Para la *región* y el *identificador de cuenta*, proporcione los que alojan su clúster de Amazon MSK.
  - Para *mskClusterName*, proporcione el nombre de su clúster de Amazon MSK.
  - Para *cluster-uuid*, proporcione el UUID del ARN de su clúster de Amazon MSK.
  - Para *mskTopicName*, proporcione el nombre del tema de Kafka.
  - Para *mskGroupName*, proporcione el nombre de su grupo de consumidores de Kafka.
2. Identifique los permisos de Amazon MSK, Amazon EC2 y CloudWatch necesarios para que Lambda detecte y conecte su clúster de Amazon MSK a fin de registrar esos eventos.

La política administrada `AWSLambdaMSKExecutionRole` define de forma permisiva los permisos necesarios. Úsela en los siguientes pasos.

En un entorno de producción, evalúe `AWSLambdaMSKExecutionRole` para restringir su política de rol de ejecución según el principio de privilegio mínimo y luego escriba una política para su rol que reemplace esta política administrada.

Para obtener más información sobre el lenguaje de políticas de IAM, consulte la [documentación de IAM](#).

Ahora que ha redactado su documento de política, cree una política de IAM para poder adjuntarla a su rol. Para ello, puede utilizar la consola de la siguiente manera.

Cómo crear una política de IAM a partir de su documento de política

1. Inicie sesión en AWS Management Console Management Console y abra la consola IAM en <https://console.aws.amazon.com/iam/>.
2. En el panel de navegación de la izquierda, elija Políticas.
3. Elija Crear política.
4. En la sección Editor de políticas, seleccione la opción JSON.
5. Pegue *clusterAuthPolicy*.
6. Cuando haya terminado de agregar permisos a la política, seleccione Siguiente.
7. En la página Revisar y crear, escriba el Nombre de la política y la Descripción (opcional) para la política que está creando. Revise los Permisos definidos en esta política para ver los permisos que concede la política.
8. Elija Crear política para guardar la nueva política.

Para obtener más información, consulte [Crear políticas de IAM](#) en la documentación de IAM.

Ahora que dispone de las políticas de IAM adecuadas, cree un rol y adjúnteselas. Para ello, puede utilizar la consola de la siguiente manera.

Para crear un rol de ejecución en la consola de IAM

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. En Tipo de entidad de confianza, seleccione Servicio de AWS.
4. En Use case (Caso de uso), elija Lambda.
5. Elija Siguiente.
6. Seleccione las siguientes políticas:
  - *clusterAuthPolicy*

- `AWSLambdaMSKExecutionRole`
7. Elija Siguiente.
  8. En Nombre de rol, escriba `lambdaAuthRole` y luego elija Crear rol.

Para obtener más información, consulte [the section called “Rol de ejecución \(permisos para que las funciones accedan a otros recursos\)”](#).

## Creación de una función de Lambda para leer su tema de Amazon MSK

Cree una función de Lambda configurada para usar su rol de IAM. Puede crear la función de Lambda con la consola.

Cómo crear una función de Lambda con su configuración de autenticación

1. Abra la consola de Lambda y seleccione Crear función en el encabezado.
2. Seleccione Crear desde cero.
3. En Nombre de la función, introduzca el nombre apropiado de su elección.
4. En Tiempo de ejecución, elija la última versión compatible de Node . js para usar el código que se proporciona en este tutorial.
5. Elija Cambiar el rol de ejecución predeterminado.
6. Seleccione Usar un rol existente.
7. En Rol existente, seleccione `lambdaAuthRole`.

En un entorno de producción, normalmente es necesario agregar más políticas al rol de ejecución de la función de Lambda para procesar de forma significativa los eventos de Amazon MSK. Para obtener más información sobre cómo agregar políticas a su rol, consulte [Agregar o eliminar permisos de identidad](#) en la documentación de IAM.

## Creación de una asignación de orígenes de eventos para la función de Lambda

La asignación de orígenes de eventos de Amazon MSK proporciona al servicio Lambda la información necesaria para invocar una función de Lambda cuando se producen los eventos de Amazon MSK adecuados. Puede crear una asignación de Amazon MSK mediante la consola. Cree un desencadenador de Lambda y, a continuación, la asignación de orígenes de eventos se configurará automáticamente.



## Cómo crear un desencadenador de Lambda (y la asignación de orígenes de eventos)

1. Navegue a la página de información general de la función de Lambda.
2. En la sección de información general de la función, seleccione Agregar desencadenador en la parte inferior izquierda.
3. En el menú desplegable Seleccionar un origen, seleccione Amazon MSK.
4. No configure la autenticación.
5. Para Clúster de MSK, seleccione el nombre de su clúster.
6. En Tamaño del lote, ingrese 1. Este paso hace que esta característica sea más fácil de probar, pero no es un valor ideal en producción.
7. Para Nombre del tema, escriba un nombre para el tema de Kafka.
8. Para ID de grupo de consumidores, indique el identificador de su grupo de consumidores de Kafka.

## Actualización de la función de Lambda para leer los datos de transmisión

Lambda proporciona información sobre los eventos de Kafka a través del parámetro del método `event`. Para ver un ejemplo de estructura de un evento de Amazon MSK, consulte [the section called “Evento de ejemplo”](#). Una vez que sepa cómo interpretar los eventos de Amazon MSK reenviados por Lambda, puede modificar el código de la función de Lambda para utilizar la información que proporcionan.

Proporcione el siguiente código a la función de Lambda para registrar el contenido de un evento de Amazon MSK de Lambda con fines de prueba:

Java

SDK para Java 2.x

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de Amazon MSK con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Amazon MSK con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
      console.log('Message:', msg)
    })
  }
}
```

Puede proporcionar un código de función de Lambda con la consola.

#### Cómo actualizar el código de la función de Lambda

1. Navegue a la página de información general de la función de Lambda.
2. Elija la pestaña Código.
3. Introduzca el código proporcionado en el IDE del código fuente.
4. En la barra de navegación del código fuente, seleccione Implementar.

## Prueba de la función de Lambda para comprobar que está conectada al tema de Amazon MSK

Ahora puede comprobar si el origen de eventos está invocando o no su función de Lambda mediante la inspección de los registros de eventos de CloudWatch.

Cómo comprobar si se está invocando la función de Lambda

1. Utilice su host de administración de Kafka para generar eventos de Kafka mediante la CLI `kafka-console-producer`. Para obtener más información, consulte [Write some events into the topic](#) en la documentación de Kafka. Envíe suficientes eventos a fin de llenar el lote definido en el tamaño del lote para la asignación de orígenes de eventos especificada en el paso anterior. Si no lo hace, Lambda esperará a que se invoque más información.
2. Si la función se ejecuta, Lambda escribe lo ocurrido en CloudWatch. En la consola, navegue hasta la página de detalles de su función de Lambda.
3. Seleccione la pestaña Configuración.
4. En la barra lateral, seleccione Herramientas de monitoreo y operaciones.
5. Identifique el grupo de registro de CloudWatch en Configuración de registro. El grupo de registro debe empezar con `/aws/lambda`. Elija el enlace del grupo de registro.
6. En la consola de CloudWatch, inspeccione los eventos de registro para ver los eventos de registro que Lambda ha enviado al flujo de registro. Identifique si hay eventos de registro que contengan el mensaje de su evento de Kafka, como se muestra en la siguiente imagen. Si los hay, significa que ha conectado correctamente una función de Lambda a Amazon MSK con una asignación de orígenes de eventos de Lambda.

2020-08-06T15:06:18.861-04:00	START RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a Version: \$LATEST
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Key: mytopic-0
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Record: { topic: 'mytopic', partition: 0, offset: 38, timestamp: 1596740777633, timestampType: 'CREATE_TIME', value: 'TWVzc2FnZSAjMQ==' }
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Message: Message #1
2020-08-06T15:06:18.890-04:00	END RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a

## Uso de AWS Lambda con Amazon RDS

Puede conectar una función de Lambda a una base de datos de Amazon Relational Database Service (Amazon RDS) directamente y a través de un Amazon RDS Proxy. Las conexiones directas son útiles en escenarios sencillos y los proxies se recomiendan para la producción. Un proxy de base de datos administra un grupo de conexiones de bases de datos compartidas que permite que su función alcance niveles altos de simultaneidad sin agotar las conexiones de base de datos.

Se recomienda utilizar Amazon RDS Proxy para las funciones de Lambda que establezcan conexiones cortas y frecuentes a bases de datos o que abran y cierren una gran cantidad de conexiones a bases de datos. Para obtener más información, consulte [Conexión automática de una función de Lambda y una instancia de base de datos](#) en la Guía para desarrolladores de Amazon Relational Database Service.

### Configuración de la función para que funcione con los recursos de RDS

En la consola de Lambda, puede configurar y aprovisionar determinadas instancias de bases de datos y recursos de proxy de Amazon RDS. Para ello, vaya a las bases de datos de RDS en la pestaña Configuración. Como alternativa, también puede crear y configurar conexiones a funciones de Lambda en la consola de Amazon RDS. Al configurar una instancia de base de datos de RDS para utilizarla con Lambda, tenga en cuenta los siguientes criterios:

- Para conectarse a una base de datos, su función debe estar en la misma Amazon VPC donde se ejecuta la base de datos.
- Puede utilizar las bases de datos de Amazon RDS con los motores MySQL, MariaDB, PostgreSQL o Microsoft SQL Server.
- También puede utilizar clústeres de bases de datos de Aurora con motores MySQL o PostgreSQL.
- Debe proporcionar un secreto de Secrets Manager para la autenticación de la base de datos.
- Un rol de IAM debe otorgar permiso para utilizar el secreto y una política de confianza debe permitir que Amazon RDS asuma el rol.
- La entidad principal de IAM que usa la consola para configurar el recurso de Amazon RDS y conectarlo a su función debe tener los siguientes permisos:

#### Note

Solo necesitará los permisos de Amazon RDS Proxy si configura un Amazon RDS Proxy para administrar un grupo de conexiones de base de datos.

## Example política de permisos

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds-db:connect",
        "rds:CreateDBProxy",
        "rds:CreateDBInstance",
        "rds:CreateDBSubnetGroup",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeDBProxies",
        "rds:DescribeDBProxyTargets",
        "rds:DescribeDBProxyTargetGroups",
        "rds:RegisterDBProxyTargets",
        "rds:ModifyDBInstance",
        "rds:ModifyDBProxy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
```

```

    "Action": [
      "lambda:CreateFunction",
      "lambda:ListFunctions",
      "lambda:UpdateFunctionConfiguration"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:AttachRolePolicy",
      "iam:AttachPolicy",
      "iam:CreateRole",
      "iam:CreatePolicy"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetResourcePolicy",
      "secretsmanager:GetSecretValue",
      "secretsmanager:DescribeSecret",
      "secretsmanager:ListSecretVersionIds",
      "secretsmanager:CreateSecret"
    ],
    "Resource": "*"
  }
]
}

```

Amazon RDS cobra una tarifa por hora para los proxies en función del tamaño de la instancia de la base de datos. Consulte los [precios de los proxies de RDS](#) para obtener más información. Para obtener más información general sobre las conexiones proxy, consulte [Uso de Amazon RDS Proxy](#) en la Guía del usuario de Amazon RDS.

### Configuración de Lambda y Amazon RDS

Las consolas Lambda y Amazon RDS le ayudarán a configurar automáticamente algunos de los recursos necesarios para establecer una conexión entre Lambda y Amazon RDS.

## Conexión a una base de datos de Amazon RDS en una función de Lambda

En el siguiente ejemplo de código, se muestra cómo implementar una función de Lambda que se conecta a una base de datos de Amazon RDS. La función realiza una solicitud sencilla a la base de datos y devuelve el resultado.

Go

SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Golang v2 code here.
*/

package main

import (
    "context"
    "database/sql"
    "encoding/json"
    "fmt"
    "os"

    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
    _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
    Name string `json:"name"`
}
```



```
func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

    var dbName string = os.Getenv("DatabaseName")
    var dbUser string = os.Getenv("DatabaseUser")
    var dbHost string = os.Getenv("DBHost") // Add hostname without https
    var dbPort int = os.Getenv("Port")      // Add port number
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = os.Getenv("AWS_REGION")

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
        context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
        panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    defer db.Close()

    var sum int
    err = db.QueryRow("SELECT ?? AS sum", 3, 2).Scan(&sum)
    if err != nil {
        panic(err)
    }
    s := fmt.Sprint(sum)
    message := fmt.Sprintf("The selected sum is: %s", s)

    messageBytes, err := json.Marshal(message)
    if err != nil {
        return nil, err
    }
}
```

```
messageString := string(messageBytes)
return map[string]interface{}{
    "statusCode": 200,
    "headers":    map[string]string{"Content-Type": "application/json"},
    "body":       messageString,
}, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
public class RdsLambdaHandler implements
RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
event, Context context) {
        APIGatewayProxyResponseEvent response = new
APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                System.getenv("ProxyHostName"),
                System.getenv("Port"),
                System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
token);
                PreparedStatement statement =
connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }
        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }
    }
}
```

```
        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
            .region(Region.of(System.getenv("AWS_REGION")))
            .credentialsProvider(DefaultCredentialsProvider.create())
            .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .resourceArn(System.getenv("ProxyHostName"))
            .secretArn(System.getenv("DBUserName"))
            .database(System.getenv("DBName"))
            .sql("SELECT 'RDS IAM Authentication'")
            .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
```

```
const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
// the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

  // Create RDS Signer object
  const signer = new Signer({
    hostname: proxy_host_name,
    port: port,
    region: aws_region,
    username: db_user_name
```

```
});

// Request authorization token from RDS, specifying the username
const token = await signer.getAuthToken();
return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
  try {
    // Obtain auth token
    const token = await createAuthToken();
    const conn = await mysql.createConnection({
      host: proxy_host_name,
      user: db_user_name,
      password: token,
      database: db_name,
      ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
  }
  catch (err) {
    console.log(err);
  }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
body: string }> => {
  // Execute database flow
  const result = await dbOps();

  // Return error if result is undefined
  if (result == undefined)
    return {
      statusCode: 500,
      body: JSON.stringify(`Error with connection to DB host`)
    }

  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
  };
};
```

```
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
```



```

        'username' => getenv('DB_USERNAME'),
        'region' => getenv('AWS_REGION'),
    ];

    // Create RDS AuthTokenGenerator object
    $generator = new
AuthTokenGenerator(CredentialProvider::defaultProvider());

    // Request authorization token from RDS, specifying the username
    return $generator->createToken(
        $dbConnection['hostname'] . ':' . $dbConnection['port'],
        $dbConnection['region'],
        $dbConnection['username']
    );
}

private function getQueryResults() {
    // Obtain auth token
    $token = $this->getAuthToken();

    // Define connection configuration
    $connectionConfig = [
        'host' => getenv('DB_HOSTNAME'),
        'user' => getenv('DB_USERNAME'),
        'password' => $token,
        'database' => getenv('DB_NAME'),
    ];

    // Create the connection to the DB
    $conn = new PDO(
"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
        $connectionConfig['user'],
        $connectionConfig['password'],
        [
            PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
            PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
        ]
    );

    // Obtain the result of the query
    $stmt = $conn->prepare('SELECT ?+? AS sum');
    $stmt->execute([3, 2]);
}

```

```

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Python.

```

import json
import os
import boto3
import pymysql

# RDS settings

```

```
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']

# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'} # Ensure you have the CA bundle for SSL
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}") # Return an error message if an exception
    occurs
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
  port = ENV['Port']          # 3306
  user = ENV['DBUser']
  region = ENV['DBRegion']    # 'us-east-1'
  db_name = ENV['DBName']

  credentials = Aws::Credentials.new(
    ENV['AWS_ACCESS_KEY_ID'],
    ENV['AWS_SECRET_ACCESS_KEY'],
    ENV['AWS_SESSION_TOKEN']
  )
  rds_client = Aws::RDS::AuthTokenGenerator.new(
    region: region,
    credentials: credentials
  )

  token = rds_client.auth_token(
    endpoint: endpoint+ ':' + port,
    user_name: user,
    region: region
  )
end
```

```
begin
  conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/var/task/global-bundle.pem',
    sslverify: true,
    enable_cleartext_plugin: true
  )
  a = 3
  b = 2
  result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
  puts result
  conn.close
  {
    statusCode: 200,
    body: result.to_json
  }
rescue => e
  puts "Database connection failed due to #{e}"
end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
  http_request::{sign, SignableBody, SignableRequest, SigningSettings},
  sign::v4,
```

```
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );
};
```

```

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
        .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse:::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

```

```

let pool = sqlx::postgres::PgPoolOptions::new()
    .connect_with(opts)
    .await?;

let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
    .bind(3)
    .bind(2)
    .fetch_one(&pool)
    .await?;

println!("Result: {:?}", result);

Ok(json!({
    "statusCode": 200,
    "content-type": "text/plain",
    "body": format!("The selected sum is: {result}")
}))
}

```

## Procesamiento de las notificaciones de eventos de Amazon RDS

Puede utilizar Lambda para procesar las notificaciones de eventos desde una base de datos de Amazon RDS. Amazon RDS envía notificaciones a un tema de Amazon Simple Notification Service (Amazon SNS), que puede configurar para invocar una función Lambda. Amazon SNS ajusta el mensaje de Amazon RDS en su propio documento de evento y lo envía a su función.

Para obtener más información sobre cómo configurar una base de datos de Amazon RDS a fin de enviar notificaciones, consulte [Uso de las notificaciones de eventos de Amazon RDS](#).

### Example Mensaje de Amazon RDS en un evento de Amazon SNS

```

{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:rds-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
      "EventSource": "aws:sns",
      "Sns": {
        "SignatureVersion": "1",
        "Timestamp": "2023-01-02T12:45:07.000Z",

```



```

        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi
+tE/1+82j...65r==",
        "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "{\"Event Source\":\"db-instance\",\"Event Time\":\"2023-01-02
12:45:06.000\",\"Identifier Link\":\"https://console.aws.amazon.com/rds/home?
region=eu-west-1#dbinstance:id=dbinstanceid\",\"Source ID\":\"dbinstanceid\",\"Event ID
\":\"http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_Events.html#RDS-
EVENT-0002\",\"Event Message\":\"Finished DB Instance backup\"}",
        "MessageAttributes": {},
        "Type": "Notification",
        "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
        "TopicArn": "arn:aws:sns:us-east-2:123456789012:sns-lambda",
        "Subject": "RDS Notification Message"
    }
}
]
}

```

## Tutorial completo de Lambda y Amazon RDS

- [Uso de una función de Lambda para acceder a una base de datos de Amazon RDS](#): con la Guía del usuario de Amazon RDS, puede aprender a utilizar una función de Lambda para escribir datos en una base de datos de Amazon RDS a través de Amazon RDS Proxy. La función de Lambda leerá registros de una cola de Amazon SQS y escribirá elementos nuevos en una tabla de la base de datos siempre que se agrega un mensaje.

## Procese las notificaciones de eventos de Amazon S3 con Lambda.

Puede usar Lambda para procesar [notificaciones de eventos](#) de Amazon Simple Storage Service. Amazon S3 puede enviar un evento a una función de Lambda cuando se crea o elimina un objeto. Puede configurar las opciones de notificación en un bucket y conceder a Amazon S3 permiso para invocar una función en la política de permisos basada en recursos de la función.

### Warning

Si la función de Lambda utiliza el mismo bucket que el que la desencadena, podría ocurrir que la ejecución de la función entrara en un bucle. Por ejemplo, si el bucket activa una función cada vez que se carga un objeto y la función carga un objeto en el bucket, la función se activa indirectamente a sí misma. Para evitarlo, utilice dos buckets o configure el desencadenador para que solo se aplique a un prefijo que se utiliza para los objetos entrantes.

Amazon S3 invoca la función de [forma asíncrona](#) con un evento que contiene detalles sobre el objeto. En el siguiente ejemplo se muestra un evento que envió Amazon S3 cuando se cargó un paquete de implementación en Amazon S3.

### Example Evento de notificaciones de Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-2",
      "eventTime": "2019-09-03T19:37:27.192Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
      },
      "requestParameters": {
        "sourceIPAddress": "205.255.255.255"
      },
      "responseElements": {
        "x-amz-request-id": "D82B88E5F771F645",
        "x-amz-id-2":
"v1R7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjK1c5aLWGVHPZLj5NeC6qMa0emYBDX0o6QBU0Wo="
      }
    }
  ]
}
```

```
  },
  "s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
    "bucket": {
      "name": "amzn-s3-demo-bucket",
      "ownerIdentity": {
        "principalId": "A3I5XTEXAMAI3E"
      },
      "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
    },
    "object": {
      "key": "b21b84d653bb07b05b1e6b33684dc11b",
      "size": 1305107,
      "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
      "sequencer": "0C0F6F405D6ED209E1"
    }
  }
}
```

Para invocar la función, Amazon S3 necesita permiso de la [política basada en recursos](#) de la función. Al configurar un desencadenador de Amazon S3 en la consola de Lambda, esta modifica la política basada en recursos para permitir a Amazon S3 invocar la función si el nombre del bucket y el ID de cuenta coinciden. Si configura la notificación en Amazon S3, utilice la API de Lambda para actualizar la política. También puede utilizar la API de Lambda para conceder permisos a otra cuenta o restringir el permiso a un alias designado.

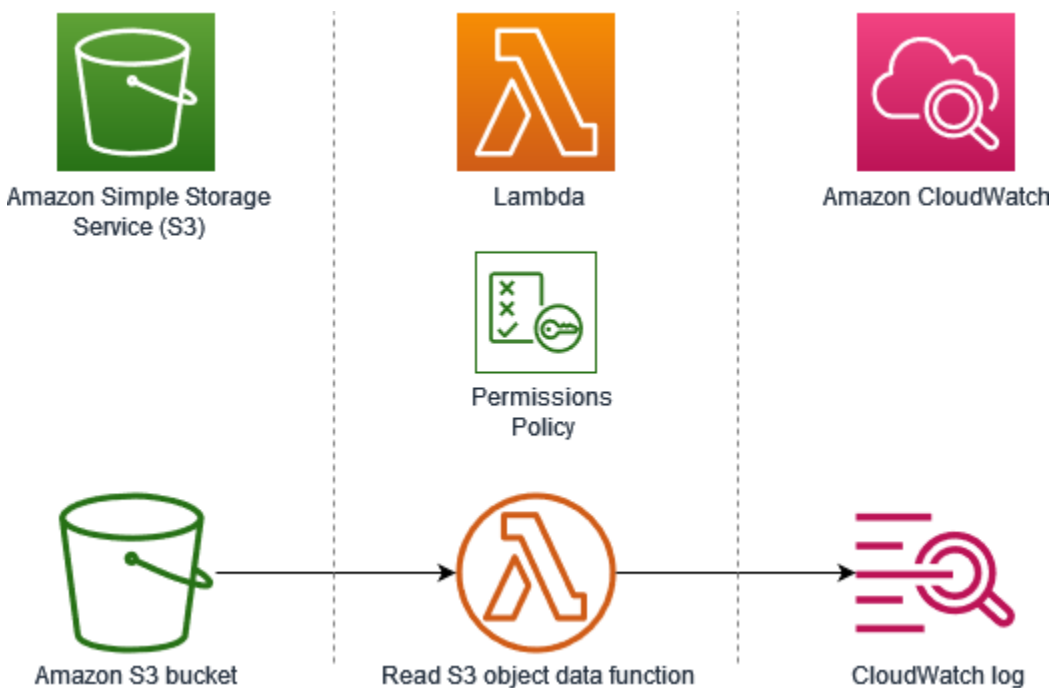
Si la función utiliza el SDK de AWS para administrar los recursos de Amazon S3, también necesita permisos de Amazon S3 en su [rol de ejecución](#).

## Temas

- [Tutorial: Uso de un desencadenador de Amazon S3 para invocar una función de Lambda](#)
- [Tutorial: Uso de un desencadenador de Amazon S3 para crear imágenes en miniatura](#)

## Tutorial: Uso de un desencadenador de Amazon S3 para invocar una función de Lambda

En este tutorial, se utiliza la consola a fin de crear una función de Lambda y configurar un desencadenador para un bucket de Amazon Simple Storage Service (Amazon S3). Cada vez que agrega un objeto al bucket de Amazon S3, la función se ejecuta y muestra el tipo de objeto en Registros de Amazon CloudWatch.



En este tutorial se muestra cómo:

1. Cree un bucket de Amazon S3.
2. Cree una función de Lambda que devuelva el tipo de objeto de los objetos en un bucket de Amazon S3.
3. Configure un desencadenador de Lambda que invoque su función cuando se carguen objetos en su bucket.
4. Pruebe su función, primero con un evento de prueba y, a continuación, con el desencadenador.

Al completar estos pasos, aprenderá a configurar una función de Lambda para que se ejecute siempre que se agreguen objetos a un bucket de Amazon S3 o se eliminen de él. Solo puede completar este tutorial mediante la AWS Management Console.

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

## Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

## Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

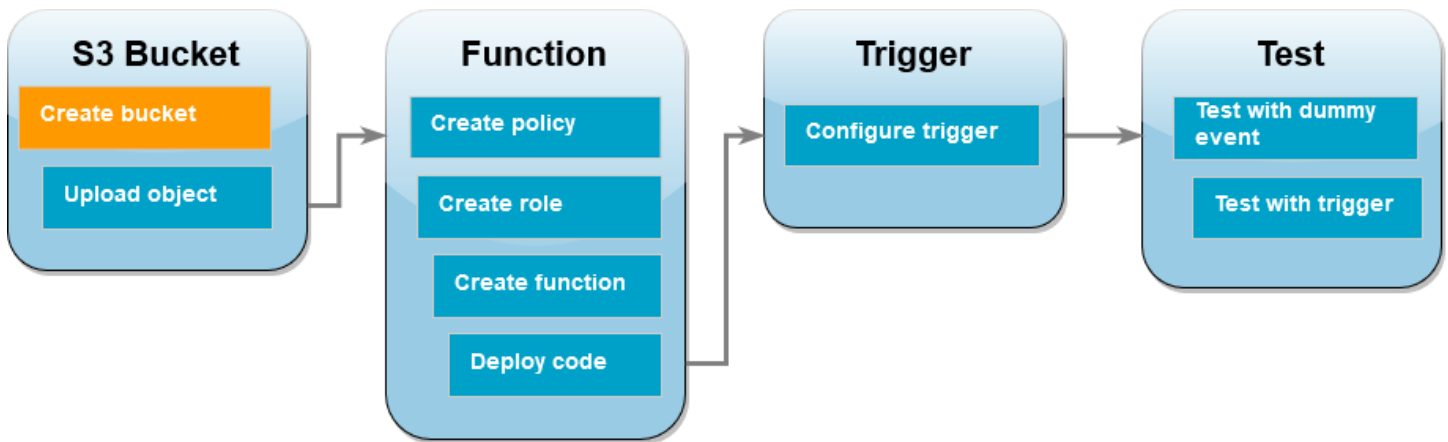
1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

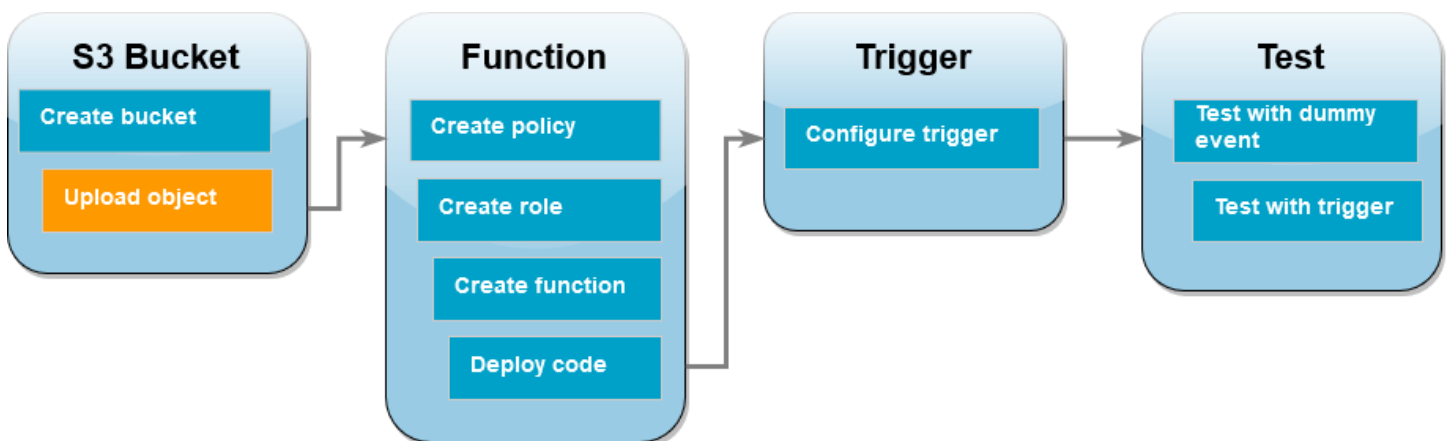
## Crear un bucket de Amazon S3



### Creación de un bucket de Amazon S3

1. Abra la [consola de Amazon S3](#) y seleccione la página Buckets.
2. Elija Crear bucket.
3. En Configuración general, haga lo siguiente:
  - a. Para el nombre del bucket, ingrese un nombre único a nivel mundial que cumpla las [reglas de nomenclatura de bucket](#) de Amazon S3. Los nombres de bucket pueden contener únicamente de letras minúsculas, números, puntos (.) y guiones (-).
  - b. En AWS Región, seleccione una región. Más adelante en el tutorial, debe crear la función de Lambda en la misma región.
4. Deje el resto de las opciones con sus valores predeterminados y seleccione Crear bucket.

### Cargar un objeto de prueba en un bucket

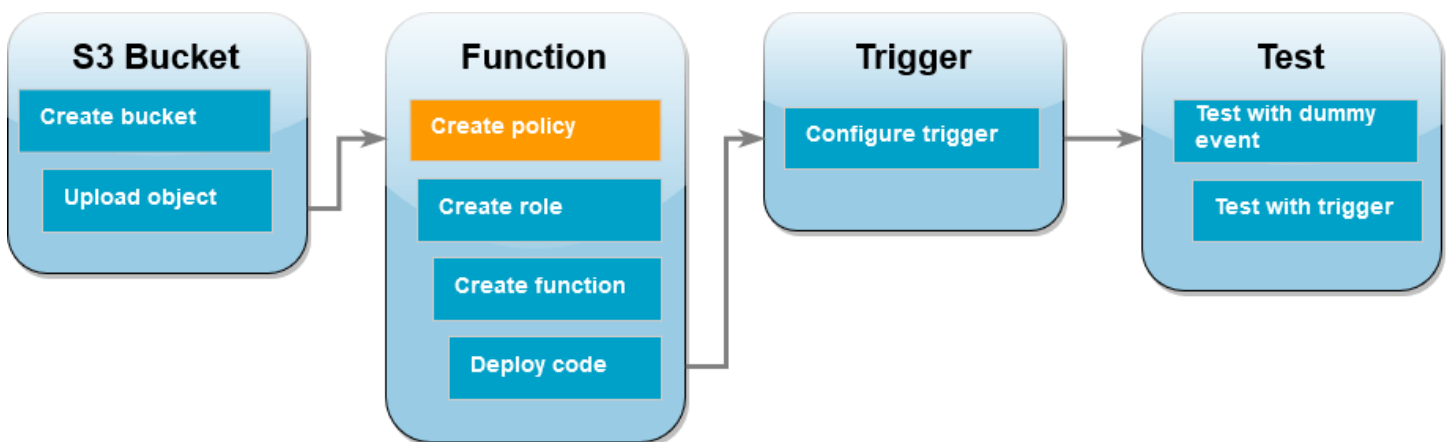


## Para cargar un objeto de prueba

1. Abra la página [Buckets](#) de la consola de Amazon S3 y elija el bucket que creó durante el paso anterior.
2. Seleccione Cargar.
3. Elija Agregar archivos y seleccione el objeto que desea cargar. Puede seleccionar cualquier archivo (por ejemplo, HappyFace . jpg).
4. Elija Abrir y, a continuación, Cargar.

Más adelante en el tutorial, probará la función de Lambda con este objeto.

## Creación de una política de permisos



Cree una política de permisos que le permita a Lambda obtener objetos de un bucket de Amazon S3 y escribir en los Registros de Amazon CloudWatch.

Para crear la política de

1. Abra la página de [Políticas \(Políticas\)](#) de la consola de IAM.
2. Seleccione Crear política.
3. Elija la pestaña JSON y pegue la siguiente política personalizada en el editor JSON.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
  
```



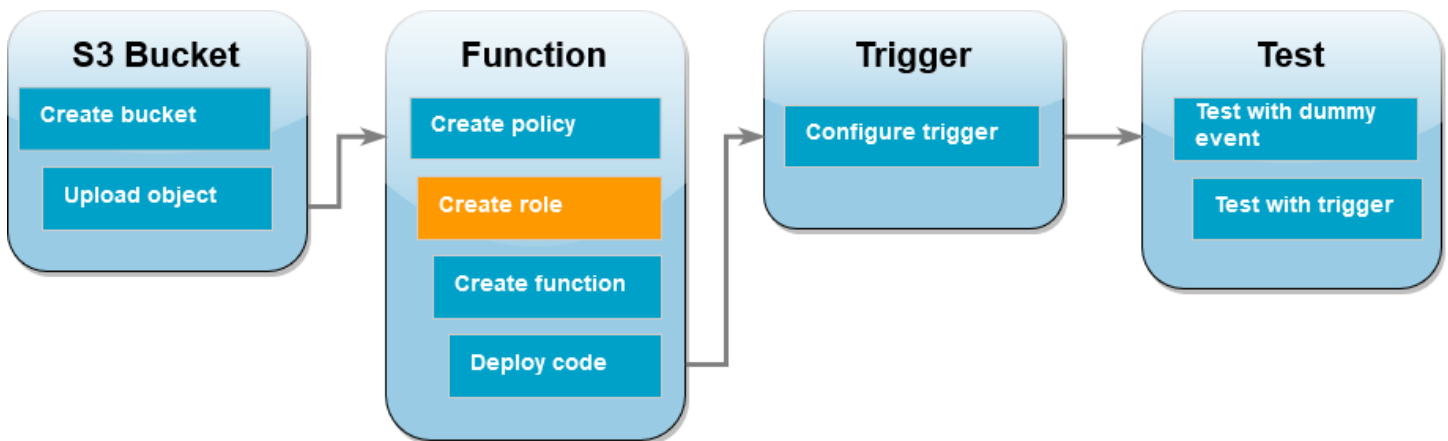
```

        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
    ],
    "Resource": "arn:aws:logs:*:*:*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": "arn:aws:s3:::*/*"
}
]
}

```

4. Elija Siguiente: Etiquetas.
5. Elija Siguiente: Revisar.
6. En Review policy (Revisar política), para el Name (Nombre) de la política, ingrese **s3-trigger-tutorial**.
7. Elija Crear política.

### Creación de un rol de ejecución

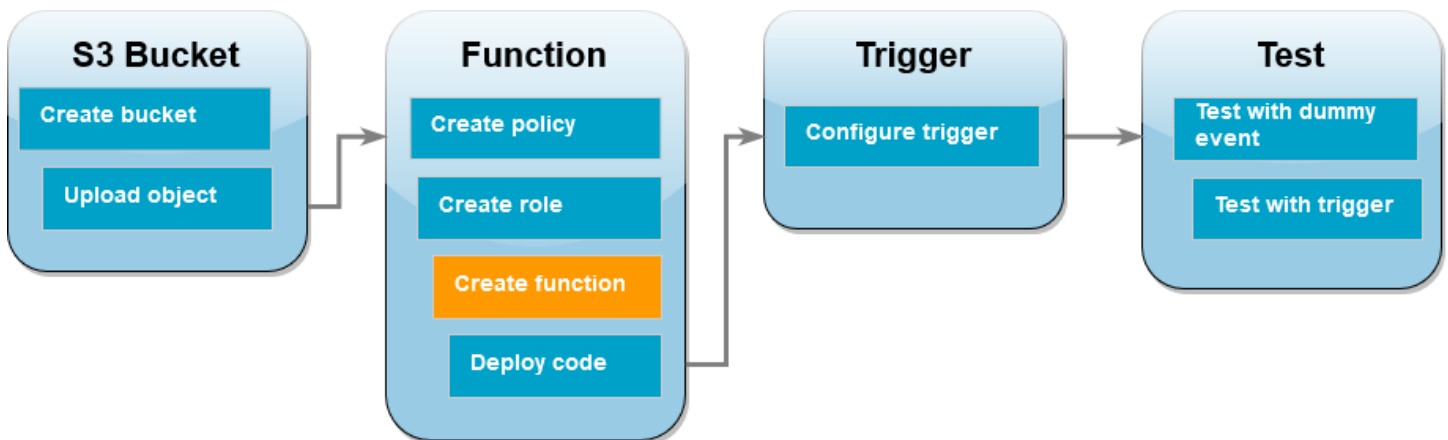


Un [rol de ejecución](#) es un rol de AWS Identity and Access Management (IAM) que concede a la función de Lambda permiso para acceder a servicios y recursos de AWS. En este paso, creará un rol de ejecución mediante la política de permisos que creó en el paso anterior.

Para crear una función de ejecución y adjuntar su política de permisos personalizada

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. Para el tipo de entidad de confianza, seleccione Servicio de AWS y, para el caso de uso, elija Lambda.
4. Elija Siguiente.
5. En el cuadro de búsqueda de políticas, escriba **s3-trigger-tutorial**.
6. En los resultados de búsqueda, seleccione la política que ha creado (s3-trigger-tutorial), y luego Next (Siguiente).
7. En Role details (Detalles del rol), introduzca **lambda-s3-trigger-role** en Role name (Nombre del rol) y, luego, elija Create role (Crear rol).

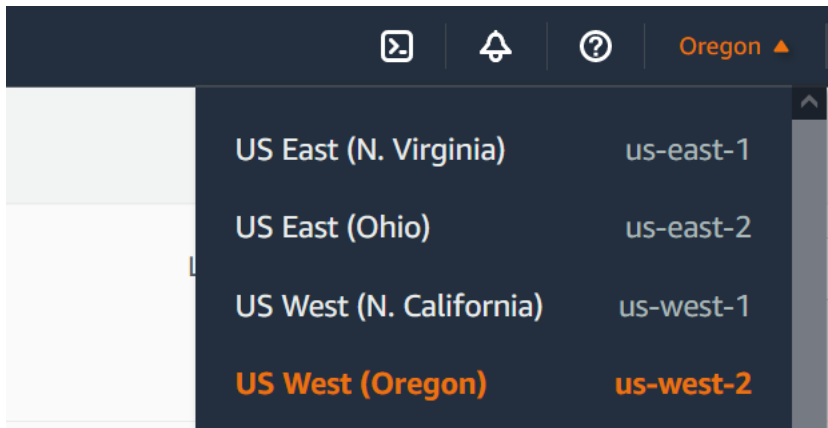
## Creación de la función de Lambda



Cree una función de Lambda en la consola con el tiempo de ejecución de Python 3.12.

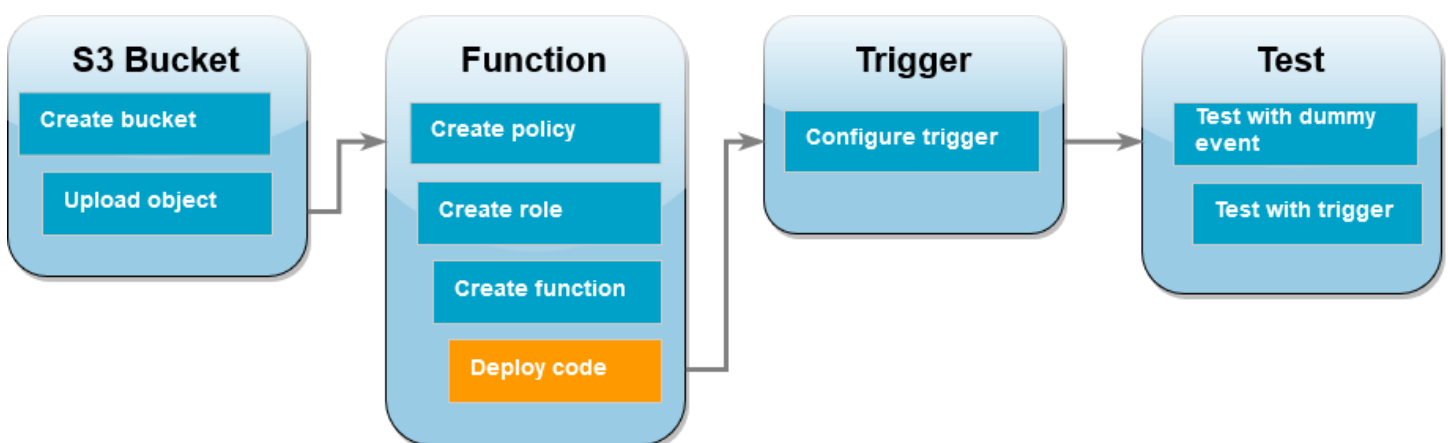
Para crear la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Asegúrese de trabajar en la misma Región de AWS en la que creó el bucket de Amazon S3. Puede cambiar la región mediante la lista desplegable de la parte superior de la pantalla.



3. Elija Crear función.
4. Elija Crear desde cero.
5. En Información básica, haga lo siguiente:
  - a. En Nombre de la función, ingrese `s3-trigger-tutorial`.
  - b. En Tiempo de ejecución, seleccione Python 3.12.
  - c. En Arquitectura, elija `x86_64`.
6. En la pestaña Cambiar rol de ejecución predeterminado, haga lo siguiente:
  - a. Amplíe la pestaña y, a continuación, elija Utilizar un rol existente.
  - b. Seleccione el `lambda-s3-trigger-role` que creó anteriormente.
7. Elija Crear función.

### Implementar el código de la función



En este tutorial, se utiliza el tiempo de ejecución de Python 3.12, pero también proporcionamos archivos de código de ejemplo para otros tiempos de ejecución. Puede seleccionar la pestaña del siguiente cuadro para ver el código del tiempo de ejecución que le interesa.


La función de Lambda recuperará el nombre de clave del objeto cargado y el nombre del bucket desde el parámetro event que recibe de Amazon S3. A continuación, la función utiliza el método [get\\_object](#) de AWS SDK for Python (Boto3) para recuperar los metadatos del objeto, incluido el tipo de contenido (tipo MIME) del objeto cargado.

Para implementar el código de la función

1. Seleccione la pestaña Python en el siguiente cuadro y copie el código.

.NET

AWS SDK for .NET

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de S3 con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be
// converted into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson

namespace S3Integration
{
```

```
public class Function
{
    private static AmazonS3Client _s3Client;
    public Function() : this(null)
    {
    }

    internal Function(AmazonS3Client s3Client)
    {
        _s3Client = s3Client ?? new AmazonS3Client();
    }

    public async Task<string> Handler(S3Event evt, ILambdaContext
context)
    {
        try
        {
            if (evt.Records.Count <= 0)
            {
                context.Logger.LogLine("Empty S3 Event received");
                return string.Empty;
            }

            var bucket = evt.Records[0].S3.Bucket.Name;
            var key =
HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

            context.Logger.LogLine($"Request is for {bucket} and {key}");

            var objectResult = await _s3Client.GetObjectAsync(bucket,
key);

            context.Logger.LogLine($"Returning {objectResult.Key}");

            return objectResult.Key;
        }
        catch (Exception e)
        {
            context.Logger.LogLine($"Error processing request -
{e.Message}");

            return string.Empty;
        }
    }
}
```

```
}  
}
```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de S3 con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
package main  
  
import (  
    "context"  
    "log"  
  
    "github.com/aws/aws-lambda-go/events"  
    "github.com/aws/aws-lambda-go/lambda"  
    "github.com/aws/aws-sdk-go-v2/config"  
    "github.com/aws/aws-sdk-go-v2/service/s3"  
)  
  
func handler(ctx context.Context, s3Event events.S3Event) error {  
    sdkConfig, err := config.LoadDefaultConfig(ctx)  
    if err != nil {  
        log.Printf("failed to load default config: %s", err)  
        return err  
    }  
    s3Client := s3.NewFromConfig(sdkConfig)  
  
    for _, record := range s3Event.Records {  
        bucket := record.S3.Bucket.Name  
        key := record.S3.Object.URLDecodedKey  
        headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
```

```
    Bucket: &bucket,
    Key:    &key,
  })
  if err != nil {
    log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
    return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
*headOutput.ContentType)
}

return nil
}

func main() {
  lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
    com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNo

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger =
        LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);
            String srcBucket = record.getS3().getBucket().getName();
            String srcKey = record.getS3().getObject().getUrlDecodedKey();

            S3Client s3Client = S3Client.builder().build();
            HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
srcKey);

            logger.info("Successfully retrieved " + srcBucket + "/" + srcKey +
" of type " + headObject.contentType());

            return "Ok";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String
bucket, String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
            .bucket(bucket)
            .key(key)
            .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```



## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

exports.handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}.
    Make sure they exist and your bucket is in the same region as this
    function.`;
    console.log(message);
    throw new Error(message);
  }
}
```

```
    }  
};
```

## Uso de un evento de S3 con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { S3Event } from 'aws-lambda';  
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';  
  
const s3 = new S3Client({ region: process.env.AWS_REGION });  
  
export const handler = async (event: S3Event): Promise<string | undefined> =>  
{  
  // Get the object from the event and show its content type  
  const bucket = event.Records[0].s3.bucket.name;  
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));  
  const params = {  
    Bucket: bucket,  
    Key: key,  
  };  
  try {  
    const { ContentType } = await s3.send(new HeadObjectCommand(params));  
    console.log('CONTENT TYPE:', ContentType);  
    return ContentType;  
  } catch (err) {  
    console.log(err);  
    const message = `Error getting object ${key} from bucket ${bucket}. Make  
sure they exist and your bucket is in the same region as this function.`;  
    console.log(message);  
    throw new Error(message);  
  }  
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de S3 con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
```

```
$bucket = $record->getBucket()->getName();
$key = urldecode($record->getObject()->getKey());

try {
    $fileSize = urldecode($record->getObject()->getSize());
    echo "File Size: " . $fileSize . "\n";
    // TODO: Implement your custom processing logic here
} catch (Exception $e) {
    echo $e->getMessage() . "\n";
    echo 'Error getting object ' . $key . ' from bucket ' .
    $bucket . '. Make sure they exist and your bucket is in the same region as
    this function.' . "\n";
    throw $e;
}
}
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de S3 con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')
```

```
s3 = boto3.client('s3')

def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']
['key'], encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they
exist and your bucket is in the same region as this function.'.format(key,
bucket))
        raise e
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de S3 con Lambda mediante Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'
```

```

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']
['key'], Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they
exist and your bucket is in the same region as this function."
    raise e
  end
end

```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante Rust.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function

```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request
from SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket =
    evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name to
exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object
key to exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;
```

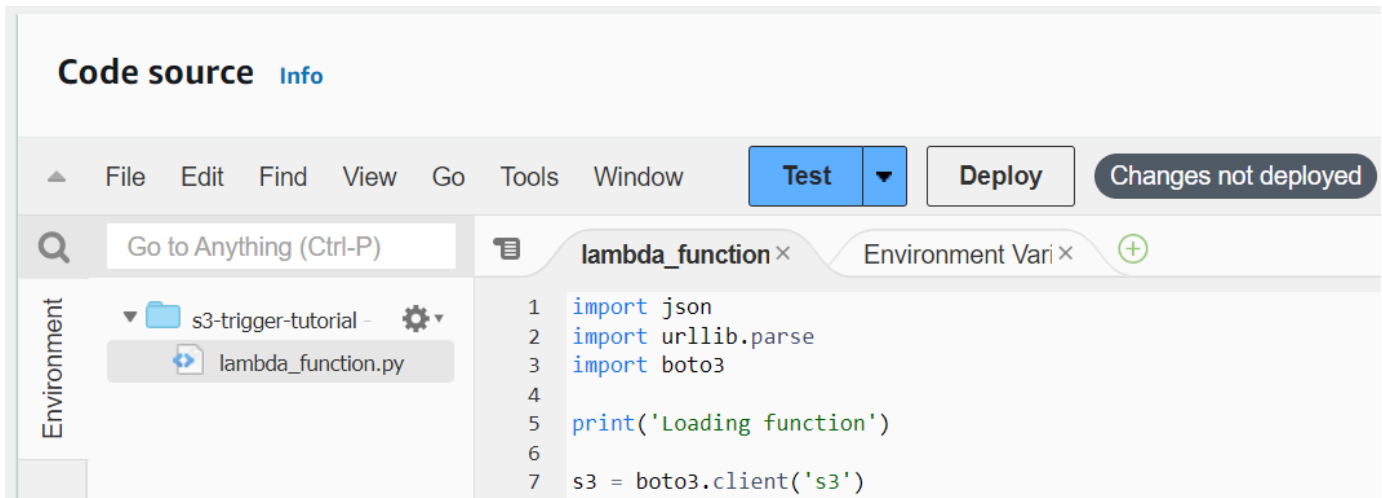
```

match s3_get_object_result {
  Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
contains a 'body' property of type ByteStream"),
  Err(_) => tracing::info!("Failure with S3 Get Object request")
}

Ok(())
}

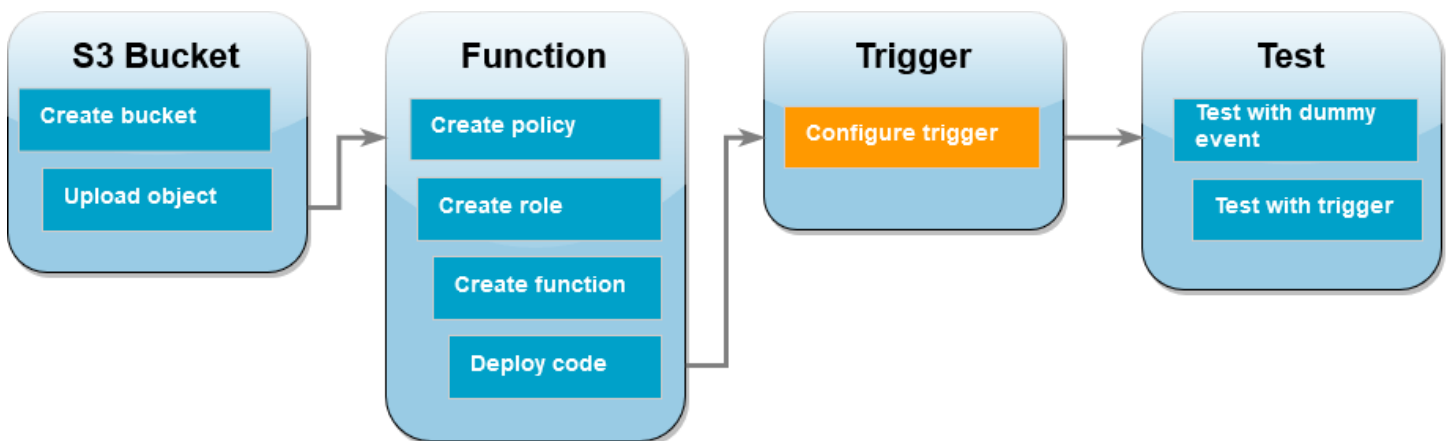
```

- En el panel Código fuente de la consola de Lambda, pegue el siguiente código en el archivo `lambda_function.py`.



- Elija Implementar.

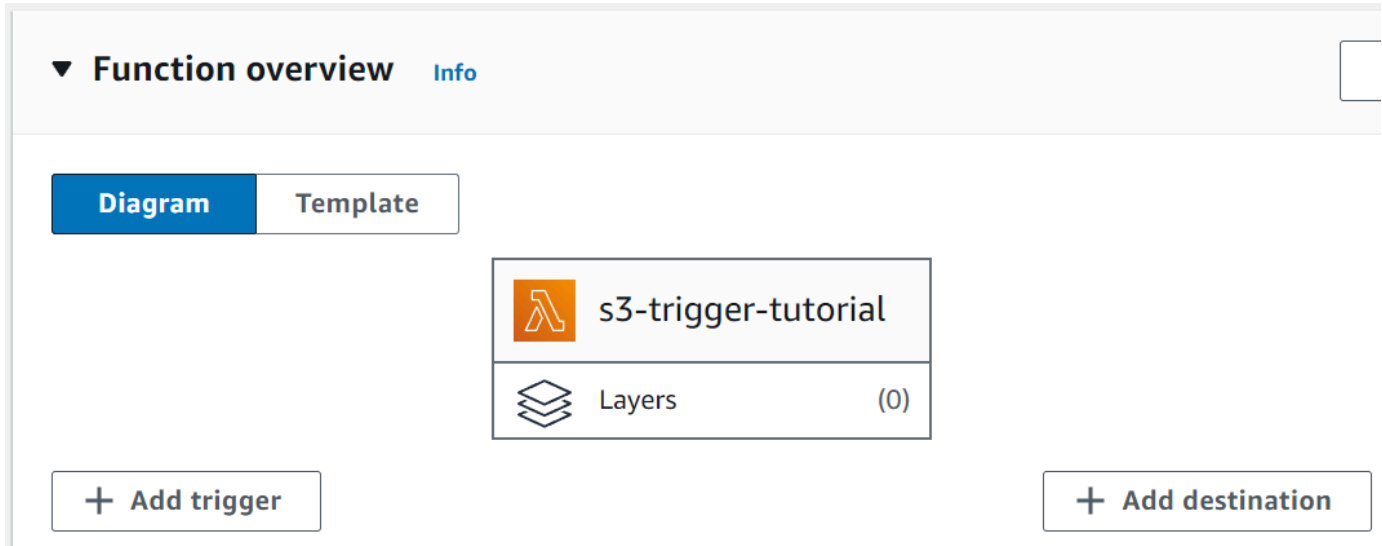
### Cree un desencadenador de Amazon S3





## Para crear el desencadenador de Amazon S3

1. En el panel Información general de la función, elija Agregar desencadenador.



2. Seleccione S3.
3. En Bucket, seleccione el bucket que creó anteriormente en el tutorial.
4. En Tipos de eventos, asegúrese de seleccionar Todos los eventos de creación de objetos.
5. En Invocación recursiva, marque la casilla de verificación para confirmar que no se recomienda utilizar el mismo bucket de Amazon S3 para la entrada y la salida.
6. Elija Añadir.

### Note

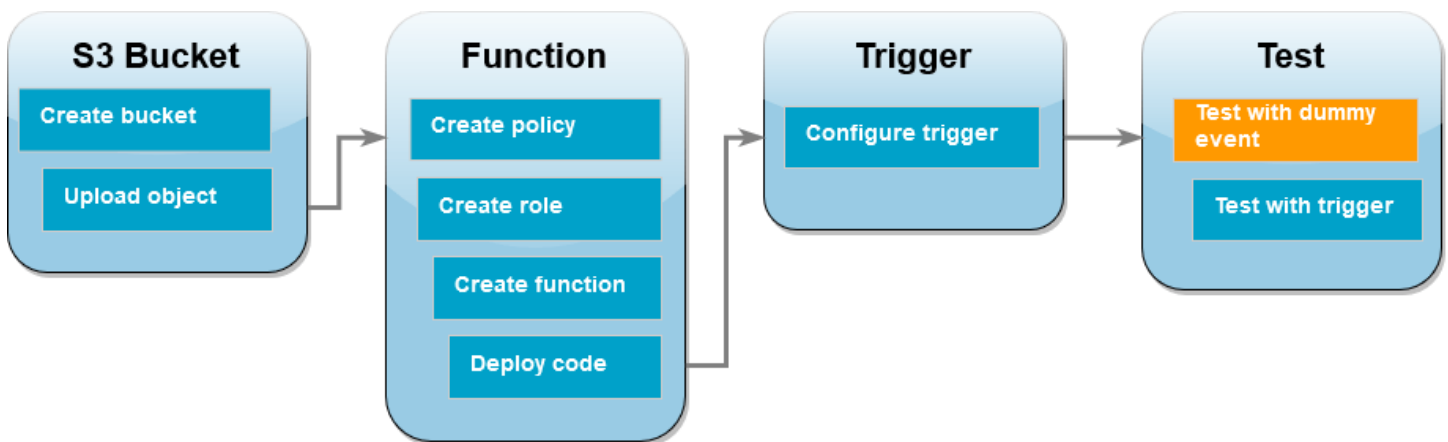
Cuando crea un desencadenador de Amazon S3 para una función de Lambda mediante la consola de Lambda, Amazon S3 configura una [notificación de eventos](#) en el bucket que especifique. Antes de configurar esta notificación de evento, Amazon S3 realiza una serie de comprobaciones para confirmar que el destino del evento existe y tiene las políticas de IAM requeridas. Amazon S3 también realiza estas pruebas en cualquier otra notificación de eventos configurada para ese bucket.

Gracias a esta comprobación, si el bucket ha configurado previamente destinos de eventos para recursos que ya no existen o para recursos que no tienen las políticas de permisos requeridas, Amazon S3 no podrá crear la nueva notificación de evento. Verá el siguiente mensaje de error que indica que no se ha podido crear el desencadenador:

An error occurred when creating the trigger: Unable to validate the following destination configurations.

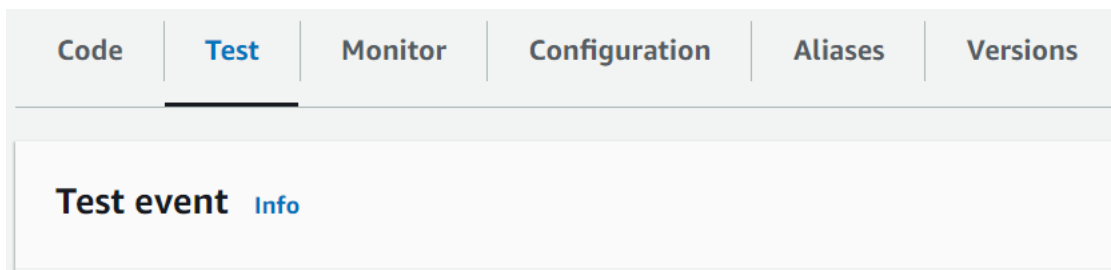
Puede ver este error si anteriormente configuró un desencadenador para otra función de Lambda con el mismo bucket y, desde entonces, ha eliminado la función o modificado sus políticas de permisos.

## Probar la función de Lambda con un evento de prueba



Para probar la función de Lambda con un evento de prueba

1. En la página de la consola de Lambda para la función, seleccione la pestaña Prueba.



2. En Nombre del evento, escriba MyTestEvent.
3. En el Evento JSON, pegue el siguiente evento de prueba. Asegúrese de reemplazar los siguientes valores:
  - Reemplace us-east-1 por la región en la que creó el bucket de Amazon S3.
  - Reemplace ambas instancias de amzn-s3-demo-bucket por el nombre de su bucket de Amazon S3.

- Reemplace `test%2Fkey` por el nombre del objeto de prueba que cargó anteriormente al bucket (por ejemplo, `HappyFace.jpg`).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmbdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

4. Seleccione Guardar.
5. Seleccione Test (Probar).
6. Si la función se ejecuta correctamente, verá un resultado similar al siguiente en la pestaña Resultados de ejecución.

## Response

```
"image/jpeg"
```

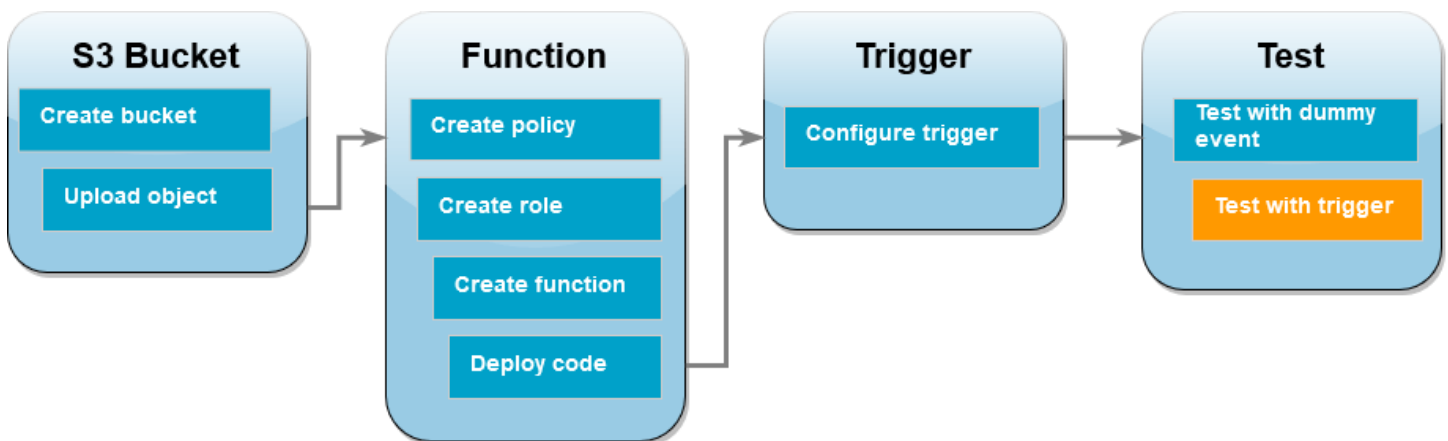
## Function Logs

```
START RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6 Version: $LATEST
2021-02-18T21:40:59.280Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    INPUT
  BUCKET AND KEY: { Bucket: 'amzn-s3-demo-bucket', Key: 'HappyFace.jpg' }
2021-02-18T21:41:00.215Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    CONTENT
  TYPE: image/jpeg
END RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6
REPORT RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6    Duration: 976.25 ms
  Billed Duration: 977 ms    Memory Size: 128 MB    Max Memory Used: 90 MB    Init
  Duration: 430.47 ms
```

## Request ID

```
12b3cae7-5f4e-415e-93e6-416b8f8b66e6
```

Probar la función de Lambda con el desencadenador de Amazon S3



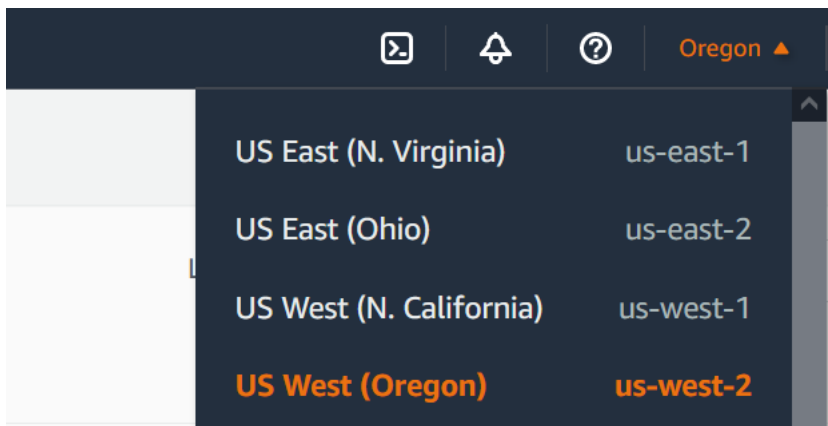
Para probar la función con el desencadenador configurado, se carga un objeto al bucket de Amazon S3 mediante la consola. Para comprobar que la función de Lambda se ha ejecutado correctamente, utilice Registros de CloudWatch para ver la salida de la función.

## Para cargar un objeto en un bucket de Amazon S3

1. Abra la página [Buckets](#) de la consola de Amazon S3 y elija el nombre del bucket que creó anteriormente.
2. Seleccione Cargar.
3. Elija Agregar archivos y utilice el selector de archivos para elegir el objeto que desee cargar. Este objeto puede ser cualquier archivo que elija.
4. Elija Abrir y, a continuación, Cargar.

## Comprobación de la invocación de la función mediante Registros de CloudWatch

1. Abra la [consola de CloudWatch](#).
2. Asegúrese de trabajar en la misma Región de AWS en la que creó la función de Lambda. Puede cambiar la región mediante la lista desplegable de la parte superior de la pantalla.



3. Elija Registros y, a continuación, Grupos de registro.
4. Elija el grupo de registro para la función (/aws/lambda/s3-trigger-tutorial).
5. En Flujos de registro, elija el flujo de registro más reciente.
6. Si su función se ha invocado correctamente en respuesta a su desencadenador de Amazon S3, verá un resultado similar al siguiente. El CONTENT TYPE que vea depende del tipo de archivo que haya subido a su bucket.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT
TYPE: image/jpeg
```

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete(Eliminar).

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

### Para eliminar el bucket de S3

1. Abra la [consola de Amazon S3](#).
2. Seleccione el bucket que ha creado.
3. Elija Eliminar.
4. Introduzca el nombre del bucket en el campo de entrada de texto.
5. Elija Delete bucket (Eliminar bucket).

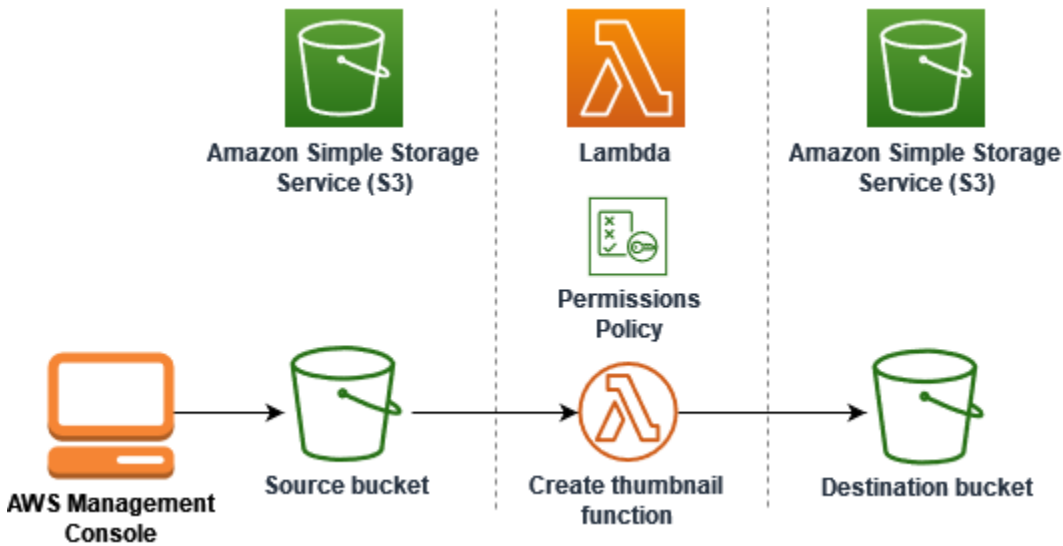
## Siguientes pasos

En [Tutorial: Uso de un desencadenador de Amazon S3 para crear imágenes en miniatura](#), el desencadenador de Amazon S3 invoca una función que crea una imagen en miniatura para cada archivo de imagen que se carga en el bucket. Este tutorial requiere un nivel moderado de conocimiento del dominio de AWS y Lambda. Demuestra cómo crear recursos usando la AWS

Command Line Interface (AWS CLI) y cómo crear un paquete de implementación de archivo .zip para la función y sus dependencias.

## Tutorial: Uso de un desencadenador de Amazon S3 para crear imágenes en miniatura

En este tutorial, se crea y configura una función de Lambda para cambiar el tamaño de las imágenes agregadas a un bucket de Amazon Simple Storage Service (Amazon S3). Al agregar un archivo de imagen al bucket, Amazon S3 invoca la función de Lambda. A continuación, la función crea una versión en miniatura de la imagen y la envía a otro bucket de Amazon S3.



Para completar este tutorial, lleve a cabo los siguientes pasos:

1. Cree buckets de origen y destino de Amazon S3 y cargue una imagen de muestra.
2. Cree una función de Lambda que cambie el tamaño de una imagen y genere una miniatura en un bucket de Amazon S3.
3. Configure un desencadenador de Lambda que invoque la función cuando se carguen objetos en el bucket de origen.
4. Pruebe la función, primero con un evento ficticio y, a continuación, cargando una imagen al bucket de origen.

Al completar estos pasos, aprenderá a utilizar Lambda para llevar a cabo una tarea de procesamiento de archivos en objetos agregados a un bucket de Amazon S3. Puede completar esta tarea mediante la AWS Command Line Interface (AWS CLI) o la AWS Management Console.

Si busca un ejemplo más sencillo para aprender a configurar un desencadenador de Amazon S3 para Lambda, puede probar el [Tutorial: Uso de un desencadenador de Amazon S3 para invocar una función de Lambda](#).

## Temas

- [Requisitos previos](#)
- [Crear dos buckets de Amazon S3](#)
- [Cargar una imagen de prueba al bucket de origen](#)
- [Creación de una política de permisos](#)
- [Creación de un rol de ejecución](#)
- [Crear el paquete de despliegue de la función](#)
- [Creación de la función de Lambda](#)
- [Configurar Amazon S3 para invocar una función](#)
- [Probar la función de Lambda con un evento de prueba](#)
- [Probar la función con el desencadenador de Amazon S3](#)
- [Eliminación de sus recursos](#)

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).



AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

### Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

### Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

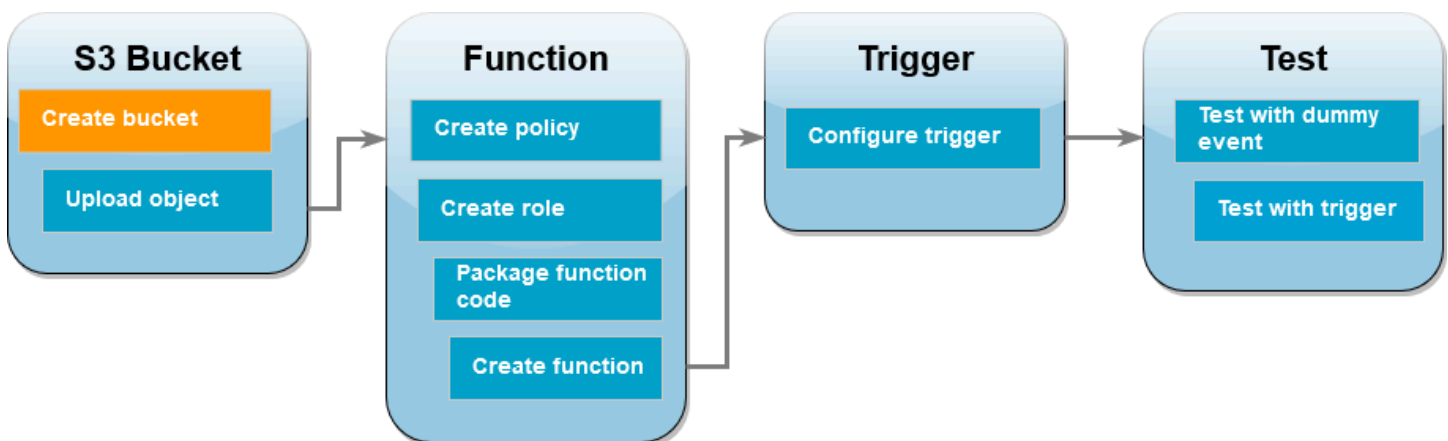
2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

Si desea utilizar la AWS CLI para completar el tutorial, instale la [versión más reciente de la AWS Command Line Interface](#).

Para el código de la función de Lambda, puede utilizar Python o Node.js. Instale las herramientas de soporte de lenguajes y un administrador de paquetes para el lenguaje que desee utilizar.

## Crear dos buckets de Amazon S3



Primero, cree dos buckets de Amazon S3. El primer bucket es el bucket de origen al que subirá las imágenes. Lambda utiliza el segundo bucket para guardar las miniaturas redimensionadas cuando se invoca la función.

## AWS Management Console

Para crear buckets de Amazon S3 (consola)

1. En la consola de Amazon S3, abra la página [Buckets](#).
2. Elija Crear bucket.
3. En Configuración general, haga lo siguiente:
  - a. Para el nombre del bucket, ingrese un nombre único a nivel mundial que cumpla las [reglas de nomenclatura de bucket](#) de Amazon S3. Los nombres de bucket pueden contener únicamente letras minúsculas, números, puntos (.) y guiones (-).
  - b. Para Región de AWS, elija la [Región de AWS](#) más cercana a su ubicación geográfica. Más adelante en el tutorial, debe crear la función de Lambda en la misma Región de AWS, por lo que debe anotar la región que eligió.
4. Deje el resto de las opciones con sus valores predeterminados y seleccione Crear bucket.
5. Repita los pasos 1 a 4 para crear el bucket de destino. En Nombre del bucket, introduzca **amzn-s3-demo-source-bucket-resized**, donde **amzn-s3-demo-source-bucket** es el nombre del bucket de origen que acaba de crear.

## AWS CLI

Para crear buckets de Amazon S3 (AWS CLI)

1. Ejecute el siguiente comando de la CLI para crear el bucket de origen. El nombre que elija para el bucket debe ser globalmente único y seguir las [reglas de nomenclatura de buckets](#) de Amazon S3. Los nombres pueden contener únicamente letras minúsculas, números, puntos (.) y guiones (-). Para region y LocationConstraint, elija la [Región de AWS](#) más cercana a su ubicación geográfica.

```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket --region us-east-1 \  
--create-bucket-configuration LocationConstraint=us-east-1
```

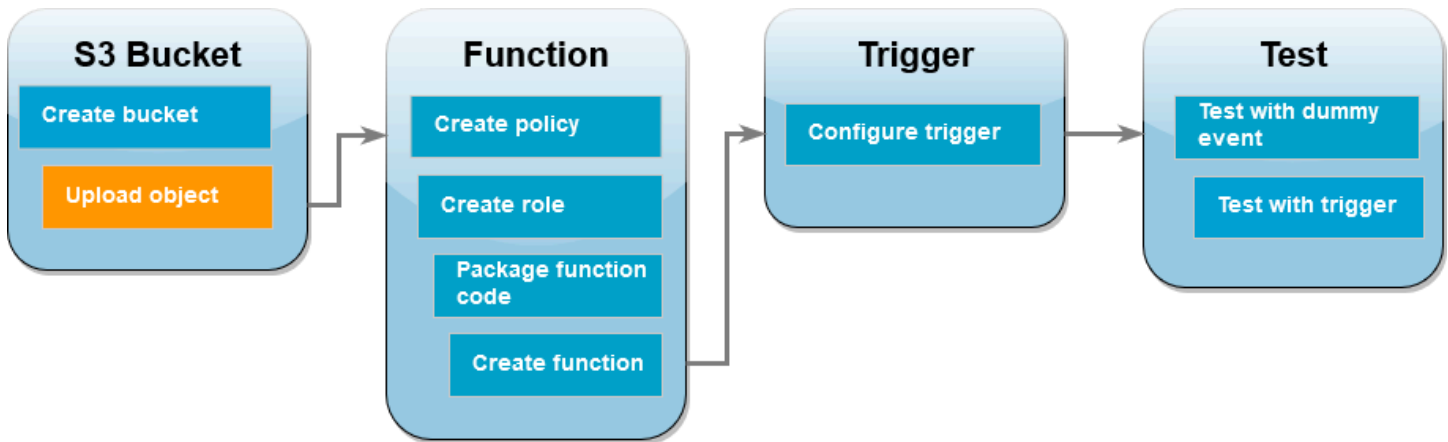
Más adelante en el tutorial, debe crear la función de Lambda en la misma Región de AWS que la del bucket de origen, por lo que debe anotar la región que eligió.

2. Ejecute el siguiente comando para crear el bucket de destino. Para el nombre del bucket, debe utilizar **amzn-s3-demo-source-bucket-resized**, donde **amzn-s3-demo-source-bucket** es el nombre del bucket de origen que creó en el paso 1. Para region

y `LocationConstraint`, elija la misma Región de AWS que usó para crear el bucket de origen.

```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket-resized --region us-east-1 \  
--create-bucket-configuration LocationConstraint=us-east-1
```

Cargar una imagen de prueba al bucket de origen



Más adelante en el tutorial, probará la función de Lambda al invocarla mediante la AWS CLI o la consola de Lambda. Para confirmar que la función se ejecuta correctamente, el bucket de origen debe contener una imagen de prueba. Esta imagen puede ser cualquier archivo JPG o PNG que elija.

## AWS Management Console

Para cargar una imagen de prueba al bucket de origen (consola)

1. En la consola de Amazon S3, abra la página [Buckets](#).
2. Seleccione el bucket de origen que creó en el paso anterior.
3. Seleccione Cargar.
4. Elija Agregar archivos y utilice el selector de archivos para elegir el objeto que desea cargar.
5. Elija Abrir y, a continuación, Cargar.

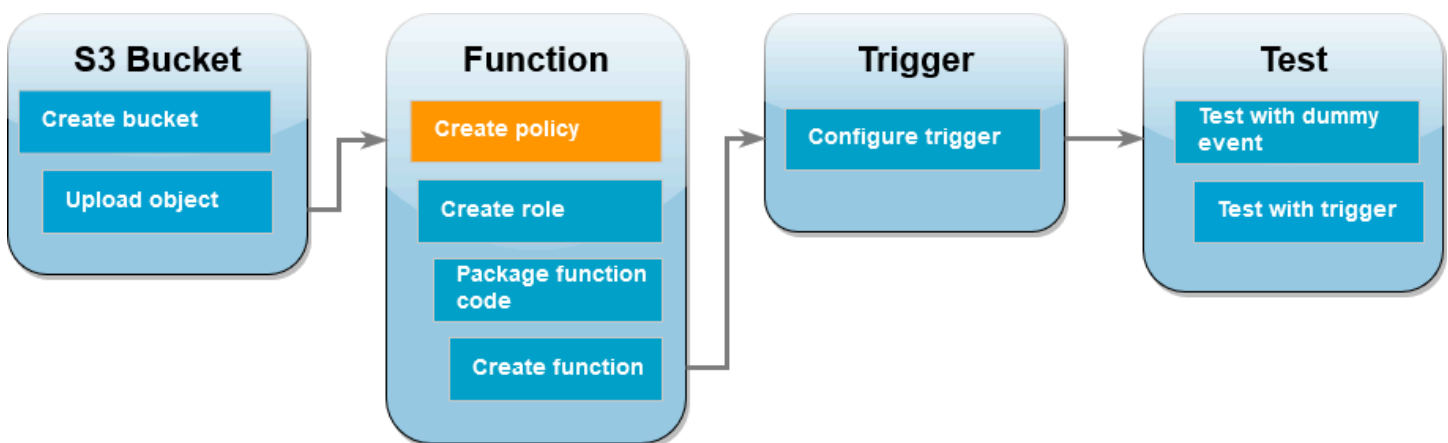
## AWS CLI

Para cargar una imagen de prueba al bucket de origen (AWS CLI)

- Desde el directorio que contiene la imagen que desea cargar, ejecute el siguiente comando de la CLI. Reemplace el parámetro `--bucket` con el nombre del bucket de origen. Para los parámetros `--key` y `--body`, utilice el nombre de archivo de la imagen de prueba.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key HappyFace.jpg --body ./HappyFace.jpg
```

## Creación de una política de permisos



El primer paso para crear la función de Lambda es crear una política de permisos. Esta política concede a la función los permisos que necesita para acceder a otros recursos de AWS. En este tutorial, la política concede permisos de lectura y escritura a Lambda para los buckets de Amazon S3 y permite que escriba en los Registros de Amazon CloudWatch.

## AWS Management Console

Para crear la política (consola)

- Abra la página de [Políticas \(Políticas\)](#) de la consola AWS Identity and Access Management (IAM).
- Elija Crear política.
- Elija la pestaña JSON y pegue la siguiente política personalizada en el editor JSON.

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogGroup",
      "logs:CreateLogStream"
    ],
    "Resource": "arn:aws:logs:*:*:*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject"
    ],
    "Resource": "arn:aws:s3:::*/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::*/*"
  }
]
}

```

4. Elija Siguiente.
5. En Detalles de política, para el Nombre de la política, ingrese **LambdaS3Policy**.
6. Elija Crear política.

## AWS CLI

Para crear la política (AWS CLI)

1. Guarde el siguiente JSON en un archivo llamado `policy.json`.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {

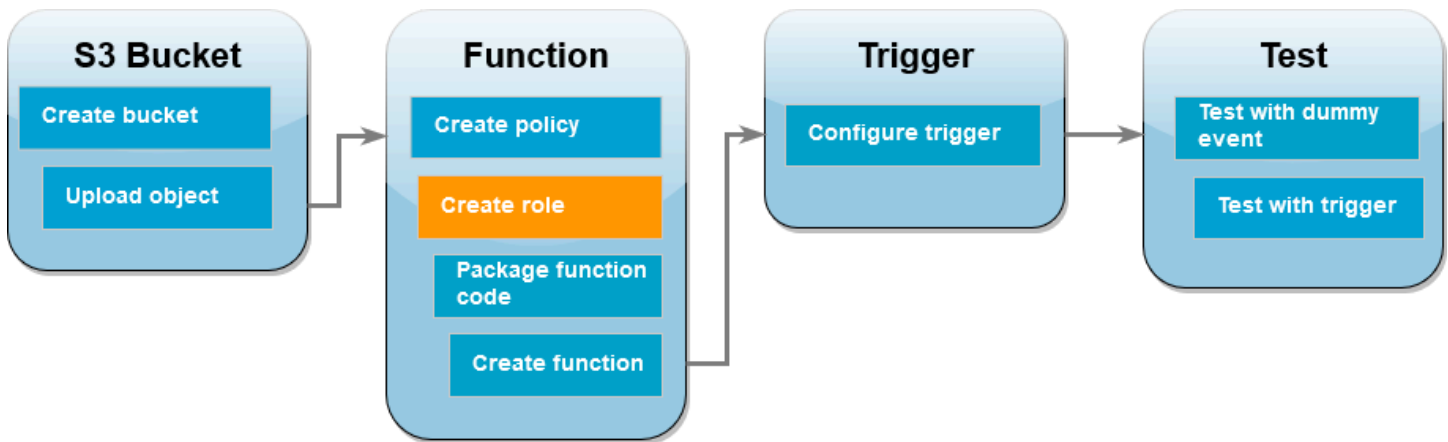
```

```
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogGroup",
      "logs:CreateLogStream"
    ],
    "Resource": "arn:aws:logs:*:*:*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject"
    ],
    "Resource": "arn:aws:s3:::*/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject"
    ],
    "Resource": "arn:aws:s3:::*/*"
  }
]
```

2. Desde el directorio en el que guardó el documento de política JSON, ejecute el siguiente comando de la CLI.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://  
policy.json
```

## Creación de un rol de ejecución



Un rol de ejecución es un rol de IAM que concede a la función de Lambda permiso para acceder a servicios y recursos de Servicios de AWS. Para conceder a la función acceso de lectura y escritura a un bucket de Amazon S3, debe adjuntar la política de permisos que creó en el paso anterior.

### AWS Management Console

Para crear un rol de ejecución y adjuntar la política de permisos (consola)

1. Abra la página [Roles](#) en la consola (IAM).
2. Elija Crear rol.
3. En Tipo de entidad de confianza, seleccione el Servicio de AWS; en Caso de uso, seleccione Lambda.
4. Elija Siguiente.
5. Agregue la política de permisos que creó en el paso anterior de la siguiente manera:
  - a. En el cuadro de búsqueda de políticas, escriba **LambdaS3Policy**.
  - b. En los resultados de búsqueda, seleccione la casilla de verificación para LambdaS3Policy.
  - c. Elija Siguiente.
6. En Detalles del rol, para el Nombre del rol, ingrese **LambdaS3Role**.
7. Elija Crear rol.



## AWS CLI

Para crear un rol de ejecución y adjuntar la política de permisos (AWS CLI)

1. Guarde el siguiente JSON en un archivo llamado `trust-policy.json`. Esta política de confianza permite a Lambda utilizar los permisos del rol al dar permiso al servicio principal `lambda.amazonaws.com` para llamar a la acción de AWS Security Token Service (AWS STS) `AssumeRole`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

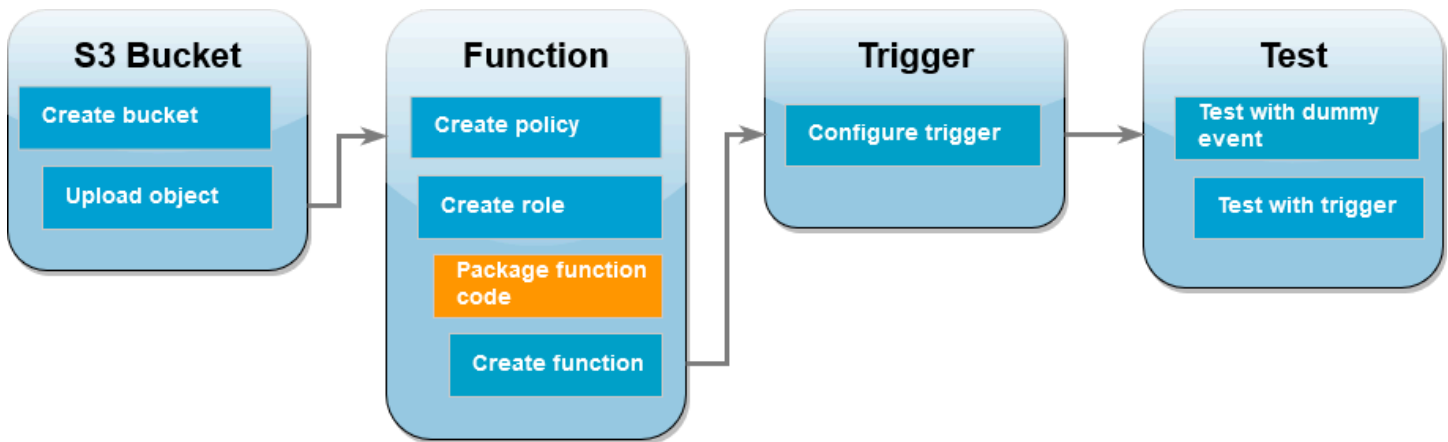
2. Desde el directorio en el que guardó el documento de política de confianza JSON, ejecute el siguiente comando de la CLI para crear el rol de ejecución.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
file://trust-policy.json
```

3. Para adjuntar la política de permisos que creó en el paso anterior, ejecute el siguiente comando de la CLI. Reemplace el número de Cuenta de AWS en el ARN de la política por su propio número de cuenta.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
arn:aws:iam::123456789012:policy/LambdaS3Policy
```

## Crear el paquete de despliegue de la función



Para crear una función, debe crear un paquete de despliegue que contenga el código y las dependencias de la función. Para esta función `CreateThumbnail`, el código de la función utiliza una biblioteca independiente para cambiar el tamaño de la imagen. Siga las instrucciones del lenguaje elegido para crear un paquete de despliegue que contenga la biblioteca requerida.

### Node.js

Para crear el paquete de despliegue (Node.js)

1. Cree un directorio llamado `lambda-s3` para el código y las dependencias de la función y desplácese hasta él.

```
mkdir lambda-s3
cd lambda-s3
```

2. Cree un nuevo proyecto de Node.js con npm. Para aceptar las opciones predeterminadas proporcionadas en la experiencia interactiva, oprima `Enter`.

```
npm init
```

3. Guarde el siguiente código de función en un archivo llamado `index.mjs`. Asegúrese de reemplazar `us-east-1` por la Región de AWS en la que creó sus propios buckets de origen y destino.

```
// dependencies
import { S3Client, GetObjectCommand, PutObjectCommand } from '@aws-sdk/client-s3';
```

```
import { Readable } from 'stream';

import sharp from 'sharp';
import util from 'util';

// create S3 client
const s3 = new S3Client({region: 'us-east-1'});

// define the handler function
export const handler = async (event, context) => {

// Read options from the event parameter and get the source bucket
console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
  const srcBucket = event.Records[0].s3.bucket.name;

// Object key may have spaces or unicode non-ASCII characters
const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
const dstBucket = srcBucket + "-resized";
const dstKey    = "resized-" + srcKey;

// Infer the image type from the file suffix
const typeMatch = srcKey.match(/\.[^\.]*$/);
if (!typeMatch) {
  console.log("Could not determine the image type.");
  return;
}

// Check that the image type is supported
const imageType = typeMatch[1].toLowerCase();
if (imageType !== "jpg" && imageType !== "png") {
  console.log(`Unsupported image type: ${imageType}`);
  return;
}

// Get the image from the source bucket. GetObjectCommand returns a stream.
try {
  const params = {
    Bucket: srcBucket,
    Key: srcKey
  };
  var response = await s3.send(new GetObjectCommand(params));
  var stream = response.Body;
```

```
// Convert stream to buffer to pass to sharp resize function.
if (stream instanceof Readable) {
  var content_buffer = Buffer.concat(await stream.toArray());

  } else {
    throw new Error('Unknown object stream type');
  }

} catch (error) {
  console.log(error);
  return;
}

// set thumbnail width. Resize will set the height automatically to maintain
// aspect ratio.
const width = 200;

// Use the sharp module to resize the image and save in a buffer.
try {
  var output_buffer = await sharp(content_buffer).resize(width).toBuffer();

} catch (error) {
  console.log(error);
  return;
}

// Upload the thumbnail image to the destination bucket
try {
  const destparams = {
    Bucket: dstBucket,
    Key: dstKey,
    Body: output_buffer,
    ContentType: "image"
  };

  const putResult = await s3.send(new PutObjectCommand(destparams));

} catch (error) {
  console.log(error);
  return;
}
```

```
console.log('Successfully resized ' + srcBucket + '/' + srcKey +  
  ' and uploaded to ' + dstBucket + '/' + dstKey);  
};
```

4. En el directorio `lambda-s3`, instale la biblioteca `sharp` con `npm`. Tenga en cuenta que la última versión de `sharp` (0.33) no es compatible con Lambda. Instale la versión 0.32.6 para completar este tutorial.

```
npm install sharp@0.32.6
```

El comando `npm install` crea un directorio `node_modules` para los módulos. Después de este paso, la estructura de directorios debería tener el siguiente aspecto.

```
lambda-s3  
|- index.mjs  
|- node_modules  
|  |- base64js  
|  |- bl  
|  |- buffer  
...  
|- package-lock.json  
|- package.json
```

5. Cree un paquete de despliegue `.zip` que contenga el código y las dependencias de la función. En Linux o macOS, ejecute el siguiente comando.

```
zip -r function.zip .
```

En Windows, utilice la utilidad de compresión que prefiera para crear un archivo `.zip`. Asegúrese de que los archivos `index.mjs`, `package.json`, `package-lock.json` y el directorio `node_modules` estén todos en la raíz del archivo `.zip`.

## Python

Para crear el paquete de despliegue (Python)

1. Guarde el código de ejemplo como un archivo denominado `lambda_function.py`.

```
import boto3
```

```
import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def lambda_handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}-resized'.format(bucket), 'resized-{}'.format(key))
```

2. En el mismo directorio en el que creó el archivo `lambda_function.py`, cree un nuevo directorio llamado `package` e instale la biblioteca [Pillow \(PIL\)](#) y AWS SDK for Python (Boto3). Si bien el tiempo de ejecución de Lambda Python incluye una versión del SDK Boto3, recomendamos agregar todas las dependencias de la función al paquete de despliegue, incluso si están incluidas en el tiempo de ejecución. Para obtener más información, consulte [Dependencias del tiempo de ejecución en Python](#).

```
mkdir package
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.12 \
--only-binary=:all: --upgrade \
pillow boto3
```

La biblioteca Pillow contiene código C y C++. Al utilizar las opciones `--platform manylinux_2014_x86_64` y `--only-binary=:all:`, pip descargará e instalará una versión de Pillow que contiene archivos binarios precompilados compatibles con el sistema operativo Amazon Linux 2. Esto garantiza que el paquete de implementación funcione en el entorno de ejecución de Lambda, independientemente del sistema operativo y la arquitectura del equipo de compilación local.

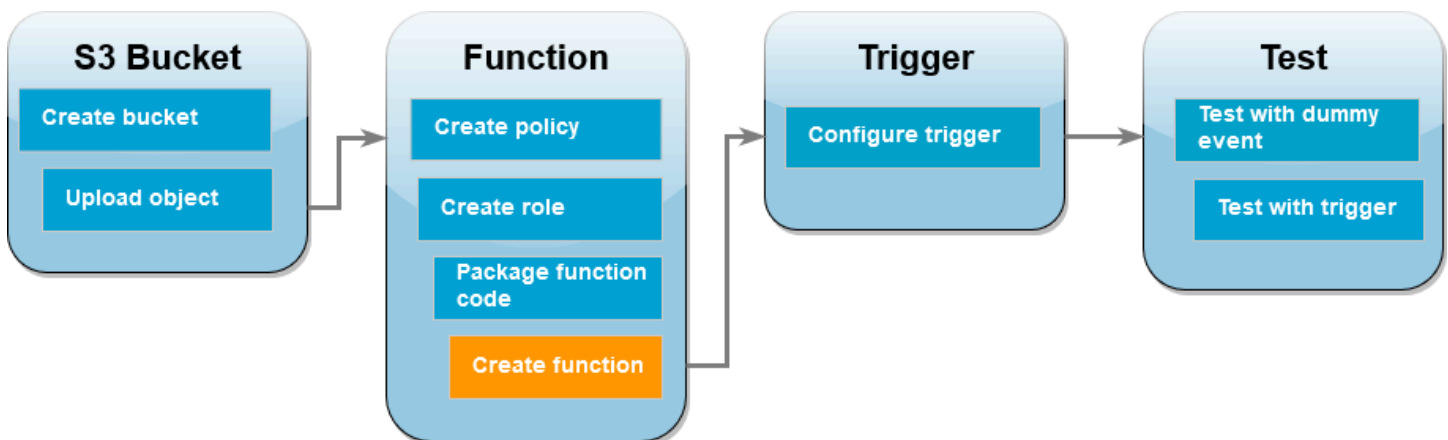
3. Cree un archivo `.zip` que contenga el código de la aplicación y las bibliotecas Pillow y Boto3. En Linux o macOS, ejecute los siguientes comandos desde la interfaz de la línea de comandos.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

En Windows, utilice la herramienta de compresión que prefiera para crear el archivo `lambda_function.zip`. Asegúrese de que el archivo `lambda_function.py` y las carpetas que contienen las dependencias estén en la raíz del archivo `.zip`.

También puede crear su paquete de implementación mediante un entorno virtual de Python. Consulte [Uso de archivos .zip para funciones de Lambda en Python](#)

## Creación de la función de Lambda



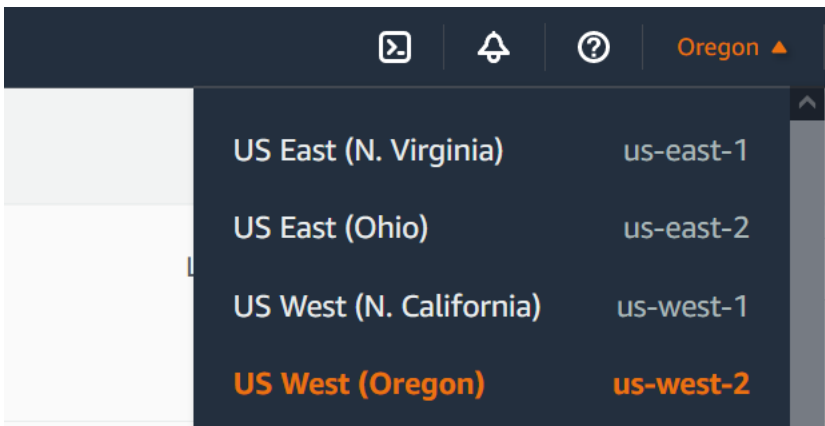
También puede crear la función de Lambda con la AWS CLI o la consola de Lambda. Siga las instrucciones del lenguaje elegido para crear la función.

## AWS Management Console

Para crear la función (consola)

Para crear una función de Lambda con la consola, primero debe crear una función básica que contenga el código “Hola, mundo”. A continuación, reemplace este código por su propio código de función mediante la carga del archivo .zip o JAR que creó en el paso anterior.

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Asegúrese de trabajar en la misma Región de AWS en la que creó el bucket de Amazon S3. Puede cambiar la región por medio de la lista desplegable de la parte superior de la pantalla.



3. Elija Crear función.
4. Elija Author from scratch (Crear desde cero).
5. Bajo Basic information (Información básica), haga lo siguiente:
  - a. En Function name (Nombre de la función), introduzca **CreateThumbnail1**.
  - b. Para el Tiempo de ejecución, elija Node.js 20.x o Python 3.12, según el lenguaje elegido para la función.
  - c. En Arquitectura, elija x86\_64.
6. En la pestaña Cambiar rol de ejecución predeterminado, haga lo siguiente:
  - a. Amplíe la pestaña y, a continuación, elija Utilizar un rol existente.
  - b. Seleccione el LambdaS3Role que creó anteriormente.
7. Elija Crear función.



Para cargar el código de la función (consola)

1. En el panel Código fuente, elija Cargar desde.
2. Elija un archivo .zip.
3. Seleccione Cargar.
4. En el selector de archivos, seleccione un archivo .zip y luego elija Abrir.
5. Seleccione Guardar.

## AWS CLI

Para crear la función (AWS CLI)

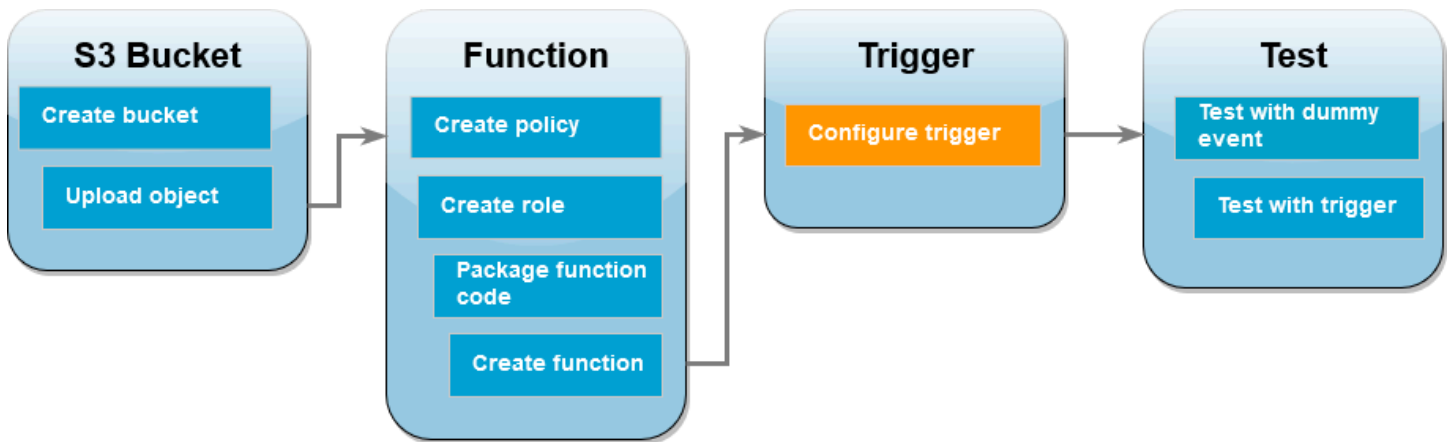
- Ejecute el comando de CLI para el lenguaje elegido. En el parámetro `role`, asegúrese de reemplazar `123456789012` por el ID de la Cuenta de AWS. En el parámetro `region`, reemplace `us-east-1` por la región en la que creó los buckets de Amazon S3.
- Para Node.js, ejecute el siguiente comando desde el directorio que contiene el archivo `function.zip`.

```
aws lambda create-function --function-name CreateThumbnail \  
--zip-file fileb://function.zip --handler index.handler --runtime nodejs20.x \  
--timeout 10 --memory-size 1024 \  
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

- Para Python, ejecute el siguiente comando desde el directorio que contiene el archivo `lambda_function.zip`.

```
aws lambda create-function --function-name CreateThumbnail \  
--zip-file fileb://lambda_function.zip --handler \  
lambda_function.lambda_handler \  
--runtime python3.12 --timeout 10 --memory-size 1024 \  
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

## Configurar Amazon S3 para invocar una función



Para que la función de Lambda se ejecute cuando carga una imagen al bucket de origen, debe configurar un desencadenador para la función. Puede configurar el desencadenador de Amazon S3 mediante la consola o la AWS CLI.

### ⚠ Important

Este procedimiento configura el bucket de Amazon S3 para invocar la función cada vez que se crea un objeto en el bucket. Asegúrese de configurar esto solo en el bucket de origen. Si la función de Lambda crea objetos en el mismo bucket que la invoca, la función se puede [invocar de forma continua en un bucle](#). Esto puede provocar que se facturen cargos imprevistos en su Cuenta de AWS.

## AWS Management Console

Para configurar el desencadenador de Amazon S3 (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija la función (CreateThumbnail).
2. Elija Add trigger (Añadir disparador).
3. Seleccione S3.
4. En Bucket, seleccione el bucket de origen.
5. En Tipos de eventos, seleccione Todos los eventos de creación de objetos.
6. En Invocación recursiva, marque la casilla de verificación para confirmar que no se recomienda utilizar el mismo bucket de Amazon S3 para la entrada y la salida. Puede

obtener más información sobre los patrones de invocación recursiva en Lambda en [Patrones recursivos que provocan funciones de Lambda descontroladas](#) en Serverless Land.

## 7. Elija Añadir.

Al crear un desencadenador mediante la consola de Lambda, Lambda crea automáticamente una [política basada en recursos](#) para conceder permiso al servicio que seleccione para invocar la función.

## AWS CLI

Para configurar el desencadenador de Amazon S3 (AWS CLI)

1. Para que el bucket de origen de Amazon S3 invoque la función al agregar un archivo de imagen, primero debe configurar los permisos de la función mediante una [política basada en recursos](#). Una instrucción de política basada en recursos concede permisos a otros Servicios de AWS para invocar la función. Para conceder permisos a Amazon S3 para invocar la función, ejecute el siguiente comando de la CLI. Asegúrese de reemplazar el parámetro `source-account` por su propio ID de Cuenta de AWS y de utilizar su propio nombre de bucket de origen.

```
aws lambda add-permission --function-name CreateThumbnail \  
--principal s3.amazonaws.com --statement-id s3invoke --action \  
"lambda:InvokeFunction" \  
--source-arn arn:aws:s3:::amzn-s3-demo-source-bucket \  
--source-account 123456789012
```

La política que defina con este comando permite a Amazon S3 invocar la función solo cuando se lleva a cabo una acción en el bucket de origen.

### Note

Si bien los nombres de los buckets de Amazon S3 son globalmente únicos, cuando se utilizan políticas basadas en recursos, se recomienda aclarar que el bucket debe pertenecer a la cuenta. Esto sucede porque si elimina un bucket, es posible que otra Cuenta de AWS cree un bucket con el mismo Nombre de recurso de Amazon (ARN).

2. Guarde el siguiente JSON en un archivo llamado `notification.json`. Cuando se aplica al bucket de origen, este JSON configura el bucket para enviar una notificación a la función

de Lambda cada vez que se agrega un objeto nuevo. Reemplace el número de Cuenta de AWS y la Región de AWS en el ARN de la función de Lambda por su número de cuenta y su región.

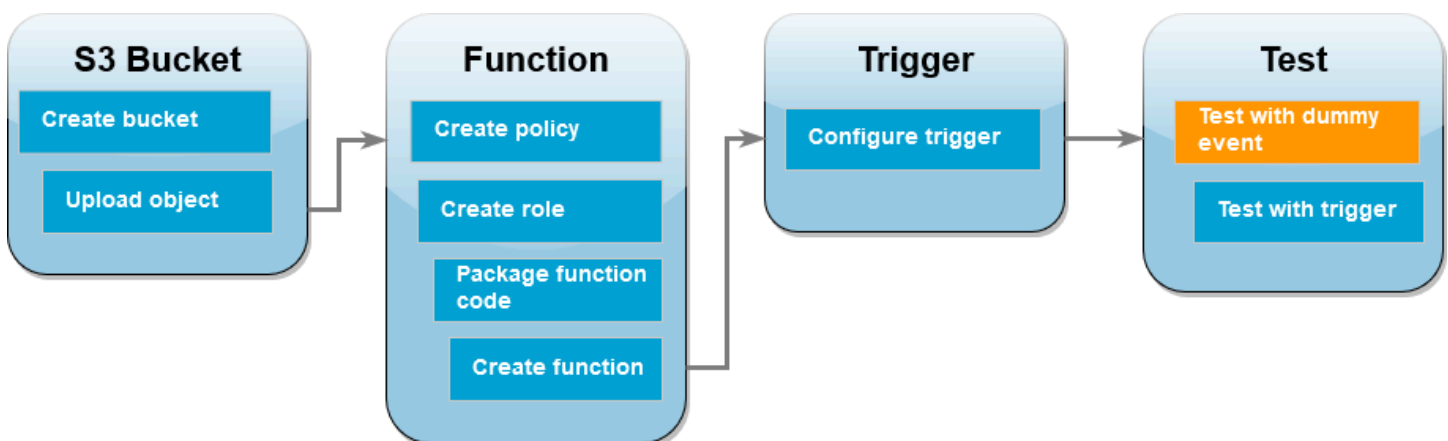
```
{
  "LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Ejecute el siguiente comando de la CLI para aplicar la configuración de notificación del archivo JSON que creó al bucket de origen. Reemplace `amzn-s3-demo-source-bucket` por el nombre del bucket de origen.

```
aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-source-
bucket \
--notification-configuration file://notification.json
```

Para obtener más información sobre el comando `put-bucket-notification-configuration` y la opción `notification-configuration`, consulte [put-bucket-notification-configuration](#) en la Referencia de comandos de la CLI AWS.

Probar la función de Lambda con un evento de prueba



Antes de probar toda la configuración al agregar un archivo de imagen al bucket de origen de Amazon S3, debe comprobar que la función de Lambda funciona correctamente al invocarla con un evento ficticio. Un evento en Lambda es un documento con formato JSON que contiene datos para que una función los procese. Cuando Amazon S3 invoca la función, el evento enviado a la función contiene información como el nombre del bucket, el ARN del bucket y la clave del objeto.

## AWS Management Console

Para probar la función de Lambda con un evento ficticio (consola)

1. Abra la [página Funciones](#) de la consola de Lambda y elija la función (CreateThumbnail).
2. Elija la pestaña Prueba.
3. Para crear el evento de prueba, en el panel Evento de prueba, haga lo siguiente:
  - a. En Acción del evento de prueba, seleccione Crear nuevo evento.
  - b. En Nombre del evento, escriba **myTestEvent**.
  - c. En Plantilla, seleccione S3 Put.
  - d. Reemplace los valores de los siguientes parámetros por sus propios valores.
    - En `awsRegion`, reemplace `us-east-1` por la Región de AWS en la que creó el bucket de Amazon S3.
    - En `name`, reemplace `amzn-s3-demo-bucket` por el nombre del bucket de origen de Amazon S3.
    - En `key`, reemplace `test%2Fkey` por el nombre del objeto de prueba que cargó al bucket de origen en el paso [Cargar una imagen de prueba al bucket de origen](#).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
```

```

    "sourceIPAddress": "127.0.0.1"
  },
  "responseElements": {
    "x-amz-request-id": "EXAMPLE123456789",
    "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
  },
  "s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "testConfigRule",
    "bucket": {
      "name": "amzn-s3-demo-bucket",
      "ownerIdentity": {
        "principalId": "EXAMPLE"
      },
      "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    "object": {
      "key": "test%2Fkey",
      "size": 1024,
      "eTag": "0123456789abcdef0123456789abcdef",
      "sequencer": "0A1B2C3D4E5F678901"
    }
  }
}
]
}

```

- e. Seleccione Guardar.
4. En el panel Evento de prueba, seleccione Prueba.
5. Para comprobar que la función creó una versión redimensionada de la imagen y la almacenó en el bucket de Amazon S3 de destino, haga lo siguiente:
  - a. En la consola de Amazon S3, abra la página [Buckets](#).
  - b. Elija el bucket de destino y confirme que el archivo redimensionado aparezca en el panel Objetos.

## AWS CLI

Para probar la función de Lambda con un evento ficticio (AWS CLI)

1. Guarde el siguiente JSON en un archivo llamado `dummyS3Event.json`. Reemplace los valores de los siguientes parámetros por sus propios valores:
  - En `awsRegion`, reemplace `us-east-1` por la Región de AWS en la que creó el bucket de Amazon S3.
  - En `name`, reemplace `amzn-s3-demo-bucket` por el nombre del bucket de origen de Amazon S3.
  - En `key`, reemplace `test%2Fkey` por el nombre del objeto de prueba que cargó al bucket de origen en el paso [Cargar una imagen de prueba al bucket de origen](#).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          }
        }
      }
    }
  ]
}
```

```

        "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    "object": {
        "key": "test%2Fkey",
        "size": 1024,
        "eTag": "0123456789abcdef0123456789abcdef",
        "sequencer": "0A1B2C3D4E5F678901"
    }
}
]
}

```

- Desde el directorio en el que guardó el archivo `dummyS3Event.json`, invoque la función mediante la ejecución del siguiente comando de la CLI. Este comando invoca la función de Lambda de forma sincrónica al definir `RequestResponse` como valor del parámetro de tipo de invocación. Para obtener más información sobre la invocación sincrónica y asíncrona, consulte [Invocación de funciones de Lambda](#).

```

aws lambda invoke --function-name CreateThumbnail \
  --invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
  --payload file://dummyS3Event.json outputfile.txt

```

La opción `cli-binary-format` es obligatoria si utiliza la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [las opciones globales de la línea de comandos admitidas de AWS CLI](#).

- Compruebe que la función haya creado una versión en miniatura de la imagen y la haya guardado en el bucket de Amazon S3 de destino. Ejecute el siguiente comando de la CLI, para ello reemplace `amzn-s3-demo-source-bucket-resized` por el nombre del bucket de destino.

```

aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized

```

Debería ver un resultado similar a este. El parámetro `Key` muestra el nombre del archivo de imagen redimensionado.

```

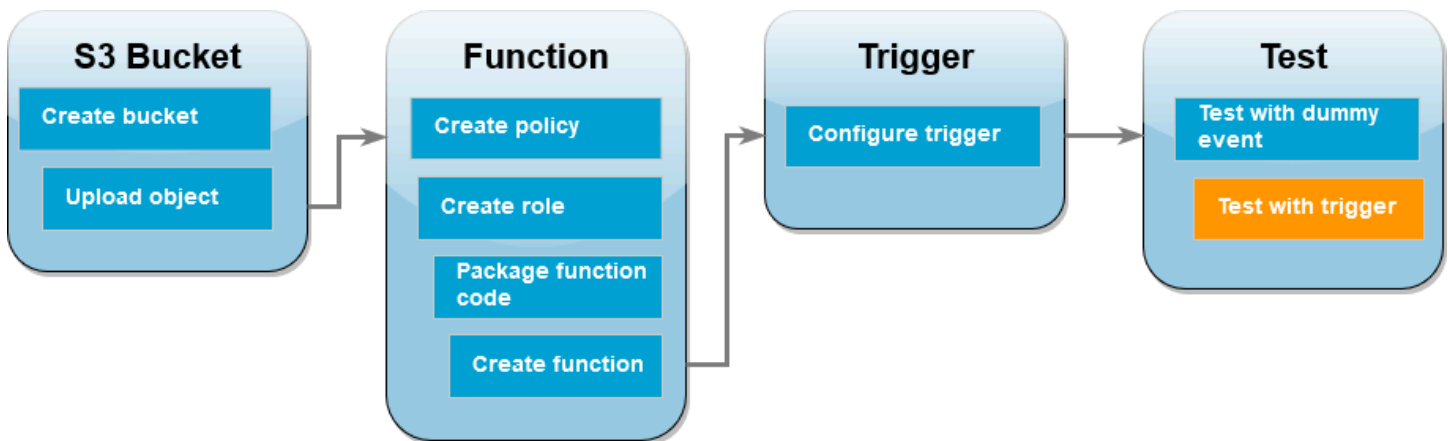
{
  "Contents": [

```



```
{
  "Key": "resized-HappyFace.jpg",
  "LastModified": "2023-06-06T21:40:07+00:00",
  "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",
  "Size": 2633,
  "StorageClass": "STANDARD"
}
]
```

## Probar la función con el desencadenador de Amazon S3



Ahora que confirmó que la función de Lambda se ejecuta correctamente, puede probar toda la configuración al agregar un archivo de imagen al bucket de origen de Amazon S3. Al agregar la imagen al bucket de origen, la función de Lambda se debe invocar automáticamente. La función crea una versión redimensionada del archivo y la almacena en el bucket de destino.

### AWS Management Console

Para probar la función de Lambda mediante el desencadenador de Amazon S3 (consola)

1. Para cargar una imagen en el bucket de Amazon S3, haga lo siguiente:
  - a. Abra la página [Buckets](#) de la consola de Amazon S3 y elija el bucket de origen.
  - b. Seleccione Cargar.
  - c. Elija Agregar archivos y utilice el selector de archivos para elegir el archivo de imagen que desea cargar. El objeto de imagen puede ser cualquier archivo .jpg o .png.
  - d. Elija Abrir y, a continuación, Cargar.

2. Compruebe que Lambda haya guardado una versión redimensionada del archivo de imagen en el bucket de destino, de la siguiente manera:
  - a. Vuelva a la página [Buckets](#) de la consola de Amazon S3 y elija el bucket de destino.
  - b. En el panel Objetos, debería ver dos archivos de imagen redimensionados, uno de cada prueba de la función de Lambda. Para descargar la imagen redimensionada, selecciona el archivo y luego elija Descargar.

## AWS CLI

Para probar la función de Lambda mediante el desencadenador de Amazon S3 (AWS CLI)

1. Desde el directorio que contiene la imagen que desea cargar, ejecute el siguiente comando de la CLI. Reemplace el parámetro `--bucket` con el nombre del bucket de origen. Para los parámetros `--key` y `--body`, utilice el nombre de archivo de la imagen de prueba. La imagen de prueba puede ser cualquier archivo `.jpg` o `.png`.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key SmileyFace.jpg --body ./SmileyFace.jpg
```

2. Compruebe que la función haya creado una versión en miniatura de la imagen y la haya guardado en el bucket de Amazon S3 de destino. Ejecute el siguiente comando de la CLI, para ello reemplace `amzn-s3-demo-source-bucket-resized` por el nombre del bucket de destino.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Si la función se ejecuta correctamente, verá un resultado similar al siguiente. El bucket de destino ahora debería contener dos archivos redimensionados.

```
{
  "Contents": [
    {
      "Key": "resized-HappyFace.jpg",
      "LastModified": "2023-06-07T00:15:50+00:00",
      "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
      "Size": 2763,
      "StorageClass": "STANDARD"
    },
  ],
}
```

```
{
  "Key": "resized-SmileyFace.jpg",
  "LastModified": "2023-06-07T00:13:18+00:00",
  "ETag": "\"ca536e5a1b9e32b22cd549e18792cdbc\"",
  "Size": 1245,
  "StorageClass": "STANDARD"
}
]
```

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete (Eliminar).

### Para eliminar la política que creó

1. Abra la página de [Políticas \(Políticas\)](#) de la consola de IAM.
2. Seleccione la política que creó (AWSLambda3Policy).
3. Elija Policy actions (Acciones de política), Delete (Eliminar).
4. Elija Delete (Eliminar).

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

## Para eliminar el bucket de S3

1. Abra la [consola de Amazon S3](#).
2. Seleccione el bucket que ha creado.
3. Elija Eliminar.
4. Introduzca el nombre del bucket en el campo de entrada de texto.
5. Elija Delete bucket (Eliminar bucket).

# Uso de Lambda con Amazon SQS

## Note

Si desea enviar datos a un destino que no sea una función de Lambda o enriquecer los datos antes de enviarlos, consulte [Amazon EventBridge Pipes](#) (Canalizaciones de Amazon EventBridge).

Puede utilizar una función de Lambda para procesar mensajes en una cola de Amazon Simple Queue Service (Amazon SQS). Las [asignaciones de orígenes de eventos](#) de Lambda son compatibles con [las colas estándar](#) y [las colas de primero en entrar, primero en salir \(FIFO\)](#).

## Temas

- [Descripción del comportamiento de sondeo y procesamiento por lotes para las asignaciones de orígenes de eventos de Amazon SQS](#)
- [Ejemplo de evento de mensaje en cola estándar](#)
- [Ejemplo de evento de mensajes de cola FIFO](#)
- [Creación y configuración de una asignación de orígenes de eventos de Amazon SQS](#)
- [Configuración del comportamiento de escalado para las asignaciones de orígenes de eventos de SQS](#)
- [Gestión de errores para un origen de eventos de SQS en Lambda](#)
- [Parámetros de Lambda para las asignaciones de origen de eventos de Amazon SQS](#)
- [Uso del filtrado de eventos con una fuente de eventos de Amazon SQS](#)
- [Tutorial: Uso de Lambda con Amazon SQS](#)
- [Tutorial: Uso de una cola entre cuentas de Amazon SQS como un origen de eventos](#)

## Descripción del comportamiento de sondeo y procesamiento por lotes para las asignaciones de orígenes de eventos de Amazon SQS

Con las asignaciones de origen de eventos de Amazon SQS, Lambda sondea la cola e invoca su función [sincrónicamente](#) con un evento. Cada evento puede contener un lote de varios mensajes de la cola. Lambda recibe estos eventos lote por lote e invoca su función una vez por lote. Cuando la función procesa correctamente un lote, Lambda elimina sus mensajes de la cola.

Cuando Lambda lee un lote, los mensajes se mantienen en la cola, pero se ocultan durante el [tiempo de espera de visibilidad](#) de la cola. Cuando la función procesa correctamente todos los mensajes en el lote, Lambda elimina sus mensajes de la cola. De forma predeterminada, si la función detecta un error al procesar un lote, todos los mensajes de ese lote se vuelven a ver en la cola después de que expire el tiempo de visibilidad. Por este motivo, el código de su función debe ser capaz de procesar el mismo mensaje varias veces sin efectos secundarios no deseados.

### Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

Para evitar que Lambda procese un mensaje varias veces, puede configurar la asignación de orígenes de eventos para incluir los [errores de los elementos del lote](#) en la respuesta de la función, o puede utilizar la acción [DeleteMessage](#) de la API para eliminar los mensajes de la cola a medida que la función de Lambda los procese correctamente.

Para obtener más información sobre los parámetros de configuración que Lambda admite para asignaciones de orígenes de eventos de SQS, consulte [the section called “Creación de una asignación de orígenes de eventos”](#).

## Ejemplo de evento de mensaje en cola estándar

Example Evento de mensaje de Amazon SQS (cola estándar)

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
```

```

        "SenderId": "AIDAIENQZJ0L023YVJ4V0",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5fBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
},
{
    "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
    "receiptHandle": "AQEBzWwaftrI0KuVm4tP+/7q1rGgNqicHq...",
    "body": "Test message.",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082650636",
        "SenderId": "AIDAIENQZJ0L023YVJ4V0",
        "ApproximateFirstReceiveTimestamp": "1545082650649"
    },
    "messageAttributes": {},
    "md5fBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
}
]
}

```

De forma predeterminada, Lambda sondea hasta 10 mensajes en su cola a la vez y envía ese lote a la función. Para evitar invocar la función con un número pequeño de registros, puede configurar el origen del evento de modo que almacene en búfer registros hasta 5 minutos mediante la definición de una ventana de lote. Antes de invocar la función, Lambda continúa el sondeo de los mensajes de la cola estándar hasta que caduque la ventana de lotes, se alcance la [cuota de tamaño de carga de invocaciones](#) o se alcance el tamaño de lote máximo configurado.

Si utiliza un intervalo de lote y la cola de SQS contiene muy poco tráfico, Lambda puede esperar hasta 20 segundos antes de invocar la función. Esto sucederá incluso si establece un intervalo de lote inferior a 20 segundos.

**Note**

En Java, es posible que se produzcan errores de puntero nulo al deserializar JSON. Esto podría deberse a la forma en que el mapeador de objetos JSON convierte las mayúsculas/minúsculas de “Records” y “EventSourceARN”.

## Ejemplo de evento de mensajes de cola FIFO

Para las colas FIFO, los registros contienen atributos adicionales relacionados con la deduplicación y la secuenciación.

Example Evento de mensaje de Amazon SQS (cola de FIFO)

```
{
  "Records": [
    {
      "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
      "receiptHandle": "AQEBBX8nesZEXmkhsmZeyIE8iQAMig7qw...",
      "body": "Test message.",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1573251510774",
        "SequenceNumber": "18849496460467696128",
        "MessageGroupId": "1",
        "SenderId": "AIDAI023YVJENQZJOL4V0",
        "MessageDeduplicationId": "1",
        "ApproximateFirstReceiveTimestamp": "1573251510774"
      },
      "messageAttributes": {},
      "md5fBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo.fifo",
      "awsRegion": "us-east-2"
    }
  ]
}
```



# Creación y configuración de una asignación de orígenes de eventos de Amazon SQS

Para procesar los mensajes de Amazon SQS con Lambda, configure la cola con los ajustes adecuados y, a continuación, cree una asignación de origen de eventos de Lambda.

## Configurar una cola para utilizarla con Lambda

Si aún no tiene una cola de Amazon SQS, [cree una](#) que sirva como origen de eventos para la función de Lambda. A continuación, configure la cola para dar tiempo suficiente a su función de Lambda a procesar cada lote de eventos.

Para que su función tenga tiempo de procesar cada lote de registros, establezca el [tiempo de espera de visibilidad](#) de la cola de origen en al menos seis veces el [tiempo de espera que configure](#) en su función. El tiempo adicional permitirá a Lambda volver a intentar realizar los procesos en caso de que la función se vea limitada debido al procesamiento de un lote anterior.

De forma predeterminada, si Lambda detecta un error en cualquier momento del procesamiento de un lote, todos los mensajes de ese lote se vuelven a la cola. Tras el tiempo de [espera de visibilidad](#), Lambda vuelve a ver los mensajes. Puede configurar la asignación de orígenes de eventos para usar [respuestas de lote parciales](#) para devolver solo los mensajes fallidos a la cola. Si su función no puede procesar un mensaje en repetidas ocasiones, Amazon SQS podrá enviar dicho mensaje a una [cola de mensajes fallidos](#). Te recomendamos que establezcas `maxReceiveCount` la [política de redrive de la cola de origen en al menos](#) 5. Esto le da a Lambda algunas oportunidades de volver a intentarlo antes de enviar los mensajes fallidos directamente a la cola de mensajes sin salida.

## Configuración de permisos de rol de ejecución de Lambda

La política administrada de AWS, [AWSLambdaSQSQueueExecutionRole](#), contiene los permisos que Lambda necesita para poder leer de la cola de Amazon SQS. Puede [agregar esta política administrada](#) al rol de ejecución de la función.

De manera opcional, si utiliza una cola cifrada, también debe agregar el siguiente permiso a su rol de ejecución:

- [kms:Decrypt](#)

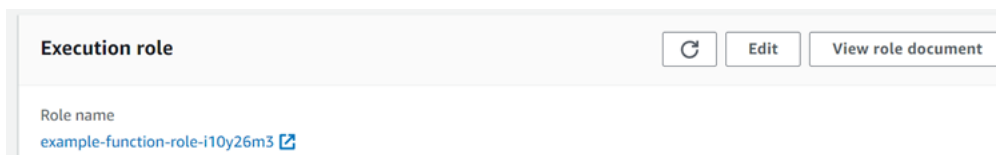
## Creación de una asignación de orígenes de eventos

Cree un mapeo de origen de eventos para indicar a Lambda que envíe elementos desde una cola a una función de Lambda. Puede crear varios mapeos de orígenes de eventos para procesar elementos de varias colas con una sola función. Cuando Lambda invoca la función objetivo, el evento puede contener múltiples elementos, dependiendo del tamaño de lote máximo configurable.

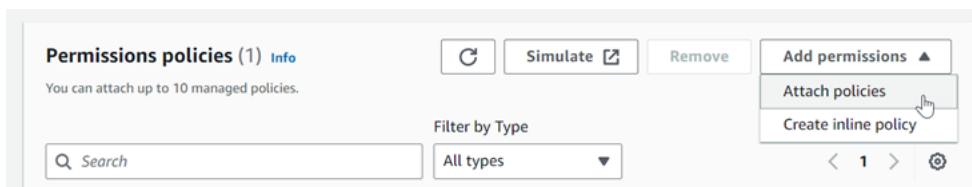
Para configurar la función para que lea desde Amazon SQS, adjunte la política administrada de AWS [AWSLambdaSQSQueueExecutionRole](#) al rol de ejecución. A continuación, cree una asignación de orígenes de eventos de SQS desde la consola mediante los siguientes pasos.

Cómo agregar permisos y crear un desencadenador

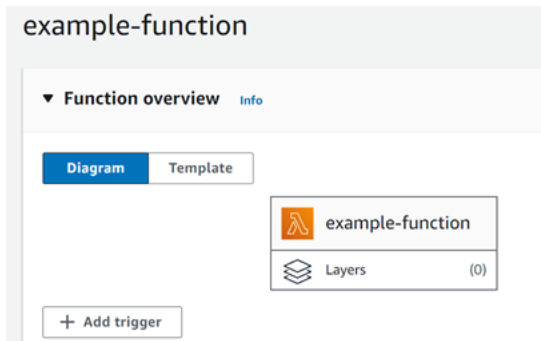
1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija la pestaña Configuración y, a continuación, elija Permisos.
4. En Nombre del rol, elija el enlace al rol de ejecución. Este enlace abre el rol en la consola de IAM.



5. Elija Agregar permisos y luego Adjuntar políticas.



6. En el campo de búsqueda, escriba `AWSLambdaSQSQueueExecutionRole`. Agregue esta política al rol de ejecución. Se trata de una política administrada de AWS que contiene los permisos que la función necesita para leer desde una cola de Amazon SQS. Para obtener más información acerca de esta política, consulte [AWSLambdaSQSQueueExecutionRole](#) en la Referencia de políticas administradas de AWS.
7. Regrese a la función en la consola de Lambda. En Descripción general de la función, elija Agregar desencadenador.



8. Elija un tipo de desencadenador.
9. Configure las opciones requeridas y luego elija Add (Agregar).

Lambda admite las siguientes opciones de configuración para los orígenes de eventos de Amazon SQS:

### Cola de SQS

La cola de Amazon SQS desde la que leer registros.

### Activar desencadenador

El estado de la asignación de origen de eventos. De forma predeterminada, la opción Enable trigger (Activar desencadenador) está seleccionada.

### Tamaño de lote

El número máximo de registros que se enviarán a la función en cada lote. Para una cola estándar, puede tratarse de hasta 10 000 registros. Para una cola FIFO, el máximo es 10. Para un tamaño de lote superior a 10, también debe establecer la ventana del lote (MaximumBatchingWindowInSeconds) a al menos 1 segundo.

Configure el [tiempo de espera de la función](#) para que disponga de tiempo suficiente para procesar todo un lote de elementos. Si elementos tardan mucho tiempo en procesarse, elija un tamaño de lote más pequeño. Un tamaño de lote amplio puede mejorar la eficiencia para cargas de trabajo que son muy rápidas o tienen mucha administración. Si configura la [simultaneidad reservada](#) en la función, establezca al menos cinco ejecuciones simultáneas para reducir la posibilidad de errores de limitación controlada cuando Lambda invoque la función.

Lambda pasa todos los registros del lote a la función en una sola llamada, siempre y cuando el tamaño total de los eventos no exceda la [cuota de tamaño de carga de invocaciones](#) para la invocación sincrónica (6 MB). Tanto Lambda como Amazon SQS generan metadatos por

cada registro. Estos metadatos adicionales se cuentan como el tamaño total de la carga útil y pueden hacer que el número total de registros enviados en un lote sea inferior al tamaño del lote configurado. Los campos de metadatos que Amazon SQS envía pueden tener una longitud variable. Para obtener más información acerca de los campos de metadatos de Amazon SQS, consulte la documentación de operación de la API [ReceiveMessage](#) en la Referencia de la API de Amazon Simple Queue Service.

## Ventana del lote

La cantidad de tiempo máxima para recopilar registros antes de invocar la función, en segundos. Esto se aplica únicamente a las colas estándar.

Si utiliza una ventana por lotes superior a 0 segundos, debe tener en cuenta el tiempo de procesamiento aumentado en el [tiempo de espera de visibilidad](#) de la cola. Recomendamos establecer el tiempo de espera de visibilidad de la cola hasta en seis veces el [tiempo de espera](#), más el valor de `MaximumBatchingWindowInSeconds`. Esto permite tiempo para que su función de Lambda procese cada lote de eventos y vuelva a intentarlo en caso de un error de limitación.

Cuando los mensajes están disponibles, Lambda comienza a procesar los mensajes en lotes. Lambda comienza a procesar 5 lotes a la vez con 5 invocaciones simultáneas de la función. Si los mensajes siguen estando disponibles, Lambda agrega hasta 300 instancias más de la función por minuto, hasta un máximo de 1000 instancias de función. Para obtener más información sobre la simultaneidad y el escalado de funciones, consulte [Escalado de la función de Lambda](#).

Para procesar más mensajes, puede optimizar la función de Lambda para obtener un mayor rendimiento. Para obtener más información, consulte [Entender cómo escala AWS Lambda con las colas estándar de Amazon SQS](#).

## Simultaneidad máxima

Número máximo de funciones simultáneas que puede invocar el origen de eventos. Para obtener más información, consulte [Configuración de la simultaneidad máxima para los orígenes de eventos de Amazon SQS](#).

## Criterios del filtro

Agregue criterios de filtrado para controlar qué eventos envía Lambda a su función para su procesamiento. Para obtener más información, consulte [Controle qué eventos envía Lambda a la función](#).

## Configuración del comportamiento de escalado para las asignaciones de orígenes de eventos de SQS

Para las colas estándar, Lambda utiliza el [sondeo largo](#) para sondear una cola hasta que se active. Cuando los mensajes están disponibles, Lambda comienza a procesar cinco lotes a la vez con cinco invocaciones simultáneas de la función. Si los mensajes siguen estando disponibles, Lambda aumenta el número de procesos que son lotes de lectura hasta 300 instancias más por minuto. El número máximo de lotes que una asignación de origen de eventos puede procesar simultáneamente es 1 000.

Para colas FIFO, Lambda envía mensajes a su función en el orden en que los recibe. Cuando envíe un mensaje a una cola FIFO, especifique un [ID de grupo de mensajes](#). Amazon SQS garantiza que los mensajes del mismo grupo se entreguen a Lambda en orden. Cuando Lambda lee los mensajes en lotes, cada lote puede contener mensajes de más de un grupo, pero se mantiene el orden de los mensajes. Si la función devuelve un error, esta realiza todos los reintentos en los mensajes afectados antes de que Lambda reciba mensajes adicionales del mismo grupo.

## Configuración de la simultaneidad máxima para los orígenes de eventos de Amazon SQS

Puede utilizar la configuración de simultaneidad máxima para controlar el comportamiento de escalado de los orígenes de eventos de SQS. La configuración de la simultaneidad máxima limita el número de instancias simultáneas de la función que puede invocar un origen de eventos de Amazon SQS. La simultaneidad máxima es una configuración a nivel de origen de eventos. Si tiene varios orígenes de eventos de Amazon SQS asignación de orígenes de eventos, cada origen de eventos puede tener una configuración de simultaneidad máxima independiente. Puede utilizar la simultaneidad máxima para evitar que una cola utilice toda la [simultaneidad reservada](#) de la función o el resto de la [cuota de simultaneidad de la cuenta](#). No hay ningún cargo por configurar la simultaneidad máxima en un origen de eventos de Amazon SQS.

Es importante destacar que la simultaneidad máxima y reservada son dos configuraciones independientes. No establezca una simultaneidad máxima superior a la reservada de la función. Si configura la simultaneidad máxima, asegúrese de que la simultaneidad reservada de la función es mayor o igual que la simultaneidad máxima total para todos los orígenes de eventos de Amazon SQS de la función. De lo contrario, Lambda podría limitar sus mensajes.

Si no se establece la simultaneidad máxima, Lambda puede escalar su origen de eventos de Amazon SQS hasta la cuota de simultaneidad total de su cuenta, que es de 1000 de forma predeterminada.

### Note

En el caso de las colas FIFO, las invocaciones simultáneas están limitadas por el número de [ID de grupos de mensajes](#) (messageGroupId) o por la configuración de simultaneidad máxima, lo que sea inferior. Por ejemplo, si tiene seis ID de grupos de mensajes y la simultaneidad máxima está establecida en 10, la función puede tener un máximo de seis invocaciones simultáneas.

Puede configurar la simultaneidad máxima en la asignación de orígenes de eventos nuevos y existentes de Amazon SQS.

Configure la simultaneidad máxima mediante la consola Lambda

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. En Function overview (Descripción general de las funciones), elija SQS. Esta abre la pestaña Configuration (Configuración).
4. Seleccione el activador de Amazon SQS y elija Edit (Editar).
5. En Maximum concurrency (Simultaneidad máxima), introduzca un número entre 2 y 1000. Para desactivar la simultaneidad máxima, deje la casilla vacía.
6. Seleccione Save (Guardar).

Configure la simultaneidad máxima mediante la AWS Command Line Interface (AWS CLI)

Utilice el comando [asignación de orígenes de eventos](#) con la opción `--scaling-config`. Ejemplo:

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --scaling-config '{"MaximumConcurrency":5}'
```

Para desactivar la simultaneidad máxima, introduzca un valor vacío para `--scaling-config`:

```
aws lambda update-event-source-mapping \  
  --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
  --scaling-config "{}"
```

Configure la simultaneidad máxima mediante la API de Lambda

Utilice las acciones de [CreateEventSourceMapping](#) o [UpdateEventSourceMapping](#) con un objeto [ScalingConfig](#).

## Gestión de errores para un origen de eventos de SQS en Lambda

Para gestionar los errores relacionados con un origen de eventos de SQS, Lambda utiliza automáticamente una estrategia de reintento con una estrategia de retroceso. También puede personalizar el comportamiento de la gestión de errores si configura la asignación de orígenes de eventos de SQS para que devuelva [respuestas parciales por lotes](#).

### Estrategia de retroceso para invocaciones fallidas

Cuando se produce un error en una invocación, Lambda intenta volver a intentar la invocación mientras implementa una estrategia de retraso. La estrategia de retroceso difiere ligeramente en función de si Lambda encontró el error debido a un error en el código de la función o a una limitación.

- Si el código de la función provocó el error, Lambda dejará de procesar y volver a intentar la invocación. Mientras tanto, Lambda reduce de manera gradual el volumen de simultaneidad asignada a su asignación de orígenes de eventos de Amazon SQS. Cuando se agote el tiempo de espera de visibilidad de la cola, el mensaje volverá a aparecer en la cola.
- Si la invocación falla debido a la limitación, Lambda retrocede gradualmente los reintentos reduciendo la cantidad de simultaneidad asignada a la asignación de orígenes de eventos de Amazon SQS. Lambda sigue reintentando enviar el mensaje hasta que la marca de tiempo del mensaje supere el tiempo límite de visibilidad de la cola, momento en el que Lambda descarta el mensaje.

### Implementación de respuestas parciales por lotes

Cuando la función Lambda detecta un error al procesar un lote, todos los mensajes de ese lote se vuelven a ver en la cola de forma predeterminada, incluidos los mensajes que Lambda procesó correctamente. Como resultado, la función puede terminar procesando el mismo mensaje varias veces.

Para evitar el reprocesamiento de todos los mensajes procesados con éxito en un lote con errores, puede configurar la asignación de orígenes de eventos para que solo se vean de nuevo los mensajes fallidos. Esto se denomina respuesta parcial por lotes. Para activar las respuestas parciales por lotes, especifique `ReportBatchItemFailures` en la acción [FunctionResponseTypes](#) cuando configure la asignación de orígenes de eventos. Esto permite que la función devuelva un éxito parcial, lo que puede ayudar a reducir el número de reintentos innecesarios en los registros.

Cuando `ReportBatchItemFailures` está activado, Lambda no [reduce verticalmente el sondeo de mensajes](#) cuando fallan las invocaciones de las funciones. Utilice `ReportBatchItemFailures`, si espera que algunos mensajes fallen y no quiere que esos errores afecten a la velocidad de procesamiento de los mensajes.

### Note

Tenga en cuenta las siguientes consideraciones al utilizar las respuestas parciales por lotes:

- Si la función genera una excepción, todo el lote se considera un error completo.
- Si utiliza esta característica con una cola FIFO, la función debe dejar de procesar los mensajes después del primer error y devolver todos los mensajes fallidos y sin procesar en `batchItemFailures`. Esto ayuda a preservar el orden de los mensajes en la cola.

Para activar los informes parciales por lotes

1. Revise las [Prácticas recomendadas para implementar las respuestas parciales por lotes](#).
2. Ejecute el siguiente comando para activar `ReportBatchItemFailures` para la función. Para recuperar el UUID de la asignación de orígenes de eventos, ejecute el comando [list-event-source-mappings](#) de la AWS CLI.

```
aws lambda update-event-source-mapping \  
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \  
--function-response-types "ReportBatchItemFailures"
```

3. Actualice el código de la función para detectar todas las excepciones y devolver los mensajes fallidos en una respuesta JSON `batchItemFailures`. La respuesta `batchItemFailures` debe incluir una lista de ID de mensajes, como valores JSON `itemIdentifier`.




Supongamos que tiene un lote de cinco mensajes con ID de mensaje `id1`, `id2`, `id3`, `id4` y `id5`. Su función procesa correctamente `id1`, `id3` y `id5`. Para hacer que los mensajes `id2` y `id4` sean de nuevo visibles en la cola, la función debe devolver la siguiente respuesta:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": "id2"
    },
    {
      "itemIdentifier": "id4"
    }
  ]
}
```

Estos son algunos ejemplos de código de función que devuelven la lista de IDs de mensajes fallidos del lote:

.NET

AWS SDK for .NET

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be
// converted into a .NET class.
[assembly:
  LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;
```

```
public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
        ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
        List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
                SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

## Go

### SDK para Go V2

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, message := range sqsEvent.Records {

        if /* Your message processing condition here */ {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": message.MessageId})
        }
    }

    sqsBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return sqsBatchResponse, nil
}
```

```
func main() {  
    lambda.Start(handler)  
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import com.amazonaws.services.lambda.runtime.Context;  
import com.amazonaws.services.lambda.runtime.RequestHandler;  
import com.amazonaws.services.lambda.runtime.events.SQSEvent;  
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,  
    SQSBatchResponse> {  
    @Override  
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context)  
    {  
  
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new  
        ArrayList<SQSBatchResponse.BatchItemFailure>();  
        String messageId = "";  
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {  
            try {  
                //process your message  
                messageId = message.getMessageId();  
            }  
        }  
    }  
}
```

```

        } catch (Exception e) {
            //Add failed message identifier to the batchItemFailures
list
            batchItemFailures.add(new
SQSBatchResponse.BatchItemFailure(messageId));
        }
    }
    return new SQSBatchResponse(batchItemFailures);
}
}

```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante JavaScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {

```

```
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

Notificación de los errores de los elementos del lote de SQS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure,
    SQSRecord } from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
    Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];


    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
```

## PHP

## SDK para PHP

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
```

```
        try {
            // Assuming the SQS message is in JSON format
            $message = json_decode($record->getBody(), true);
            $this->logger->info(json_encode($message));
            // TODO: Implement your custom processing logic here
        } catch (Exception $e) {
            $this->logger->error($e->getMessage());
            // failed processing the record
            $this->markAsFailed($record);
        }
    }
    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords SQS
records");
}
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}
```



```
for record in event["Records"]:
    try:
        # process message
    except Exception as e:
        batch_item_failures.append({"itemIdentifier":
record['messageId']})

sqs_batch_response["batchItemFailures"] = batch_item_failures
return sqs_batch_response
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
```

```
    return sqs_batch_response
  end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }
}
```

```
Ok(SqsBatchResponse {
    batch_item_failures,
})
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

Si los eventos fallidos no vuelven a la cola, consulte [¿Cómo soluciono los problemas de la función de Lambda SQS ReportBatchItemFailures?](#) en el Centro de conocimientos de AWS.

### Condiciones de éxito y fracaso

Lambda trata un lote como un éxito completo si la función devuelve cualquiera de los siguientes elementos:

- Una lista `batchItemFailures` vacía
- Una lista `batchItemFailures` nula
- Una `EventResponse` vacía
- Una `EventResponse` nula

Lambda trata un lote como un error completo si la función devuelve cualquiera de los siguientes elementos:

- Respuesta JSON no válida
- Una cadena `itemIdentifier` vacía
- Una `itemIdentifier` nula
- Un `itemIdentifier` con un mal nombre de clave
- Un valor `itemIdentifier` con un ID de mensaje que no existe

## Métricas de CloudWatch

Para determinar si la función informa correctamente de los errores de elementos por lotes, puede monitorear las métricas Amazon SQS `NumberOfMessagesDeleted` y `ApproximateAgeOfOldestMessage` en Amazon CloudWatch.

- `NumberOfMessagesDeleted` realiza un seguimiento del número de mensajes eliminados de la cola. Si cae a 0, es una señal de que la respuesta de la función no está devolviendo correctamente los mensajes fallidos.
- `ApproximateAgeOfOldestMessage` hace un seguimiento de cuánto tiempo ha permanecido el mensaje más antiguo en la cola. Un aumento brusco de esta métrica puede indicar que la función no está devolviendo correctamente los mensajes fallidos.

## Parámetros de Lambda para las asignaciones de origen de eventos de Amazon SQS

Todos los tipos de origen de eventos Lambda comparten las mismas operaciones [CreateEventSourceMapping](#) y [UpdateEventSourceMapping](#) de la API. Sin embargo, solo algunos de los parámetros se aplican a Amazon SQS.

Parámetro	Obligatoria	Predeterminado	Notas
BatchSize	N	10	Para colas estándar, el máximo es 10 000. Para colas FIFO, el máximo es 10.
Habilitado	N	true	Ninguno
EventSourceArn	Y	N/A	El ARN del flujo de datos o un consumidor de flujos
FunctionName	Y	N/A	Ninguno
FilterCriteria	N	N/A	<a href="#">Controle qué eventos envía Lambda a la función</a>

Parámetro	Obligatoria	Predeterminado	Notas
FunctionResponseTypes	N	N/A	Para permitir que la función informe de errores específicos de un lote, incluya el valor ReportBatchItemFailures en FunctionResponseTypes. Para obtener más información, consulte <a href="#">Implementación de respuestas parciales por lotes</a> .
MaximumBatchingWindowInSeconds	N	0	Ninguno
ScalingConfig	N	N/A	<a href="#">Configuración de la simultaneidad máxima para los orígenes de eventos de Amazon SQS</a>

## Uso del filtrado de eventos con una fuente de eventos de Amazon SQS

Puede utilizar el filtrado de eventos para controlar qué registros de un flujo o una cola envía Lambda a su función. Para obtener información general sobre cómo funciona el filtrado de eventos, consulte [the section called “Filtrado de eventos”](#).

Esta sección se centra en el filtrado de eventos para las fuentes de eventos de Amazon MSK.

### Temas

- [Conceptos básicos de filtrado de eventos de Amazon SQS](#)

## Conceptos básicos de filtrado de eventos de Amazon SQS

Supongamos que su cola de Amazon SQS contiene mensajes en el siguiente formato JSON.

```
{
  "RecordNumber": 0000,
  "TimeStamp": "yyyy-mm-ddThh:mm:ss",
  "RequestCode": "AAAA"
}
```

Un registro de ejemplo para esta cola tendría el siguiente aspecto.

```
{
  "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
  "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
  "body": "{\n \"RecordNumber\": 0000,\n \"TimeStamp\": \"yyyy-mm-ddThh:mm:ss\",\n\n\"RequestCode\": \"AAAA\"\n}",
  "attributes": {
    "ApproximateReceiveCount": "1",
    "SentTimestamp": "1545082649183",
    "SenderId": "AIDAIENQZJOL023YVJ4V0",
    "ApproximateFirstReceiveTimestamp": "1545082649185"
  },
  "messageAttributes": {},
  "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
  "eventSource": "aws:sqs",
  "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:my-queue",
  "awsRegion": "us-west-2"
}
```

Para filtrar en función del contenido de los mensajes de Amazon SQS, utilice la clave `body` del registro de mensajes de Amazon SQS. Supongamos que desea procesar solo los registros en los que el `RequestCode` del mensaje de Amazon SQS sea "BBBB". El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
    }
  ]
}
```

```
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "body": {
    "RequestCode": [ "BBBB" ]
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

### AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"]}]'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"]}]'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Supongamos que desea que su función procese solo los registros en los que `RecordNumber` sea un valor superior a 9999. El objeto `FilterCriteria` sería el siguiente.

```
{
  "Filters": [
    {
      "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\" : [ \">\",
9999 ] } ] } }"
    }
  ]
}
```

Para mayor claridad, este es el valor del `Pattern` del filtro ampliado en JSON no cifrado.

```
{
  "body": {
    "RecordNumber": [
      {
        "numeric": [ ">", 9999 ]
      }
    ]
  }
}
```

Puede agregar el filtro mediante la consola, la AWS CLI o una plantilla de AWS SAM.

### Console

Para agregar este filtro mediante la consola, siga las instrucciones que se indican en [Adjuntar criterios de filtro a una asignación de origen de eventos \(consola\)](#) e ingrese la siguiente cadena para los Criterios de filtro.



```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

## AWS CLI

Para crear una nueva asignación de orígenes de eventos con estos criterios de filtro mediante la AWS Command Line Interface (AWS CLI), ejecute el siguiente comando.

```
aws lambda create-event-source-mapping \
  --function-name my-function \
  --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\" : [ \">\", 9999 ] } ] } }"]}]'
```

Para agregar estos criterios de filtro a una asignación de orígenes de eventos existente, ejecute el siguiente comando.

```
aws lambda update-event-source-mapping \
  --uuid "a1b2c3d4-5678-90ab-cdef-1111EXAMPLE" \
  --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\" : [ \">\", 9999 ] } ] } }"]}]'
```

## AWS SAM

Para agregar este filtro mediante AWS SAM, agregue el siguiente fragmento a la plantilla YAML de su origen de eventos.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

Para Amazon SQS, el cuerpo del mensaje puede ser cualquier cadena. No obstante, esto puede ser problemático si los `FilterCriteria` esperan que el formato JSON del body sea válido. La situación inversa también es problemática: si el cuerpo del mensaje entrante está en formato JSON, pero los criterios de filtro esperan que el body sea una cadena sin formato, puede producirse un comportamiento no deseado.

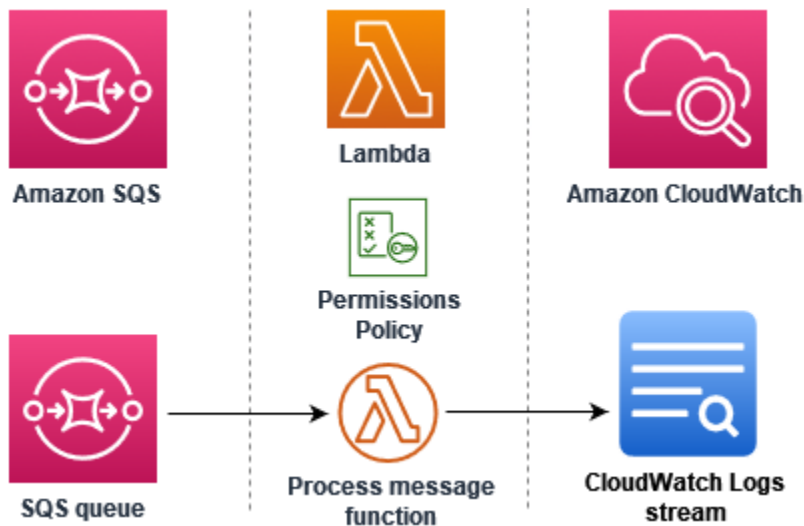
Para evitar este problema, asegúrese de que el formato del cuerpo de los `FilterCriteria` coincida con el formato esperado del body de los mensajes que recibe de la cola. Antes de filtrar los

mensajes, Lambda evalúa automáticamente el formato del cuerpo del mensaje entrante y del patrón de filtro del body. Si no coincide, Lambda elimina el mensaje. En la siguiente tabla se resume esta evaluación:

Formato del <b>body</b> del mensaje entrante	Formato del <b>body</b> del patrón de filtro	Acción resultante
Cadena sin formato	Cadena sin formato	Lambda filtra en función de los criterios de filtro.
Cadena sin formato	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
Cadena sin formato	JSON válido	Lambda elimina el mensaje.
JSON válido	Cadena sin formato	Lambda elimina el mensaje.
JSON válido	Sin patrón de filtro para las propiedades de datos	Lambda filtra (solo en las demás propiedades de metadatos) en función de los criterios de filtro.
JSON válido	JSON válido	Lambda filtra en función de los criterios de filtro.

## Tutorial: Uso de Lambda con Amazon SQS

En este tutorial, creará una función de Lambda que consuma los mensajes de una cola de [Amazon Simple Queue Service \(Amazon SQS\)](#). La función de Lambda se ejecuta cada vez que se agrega un mensaje nuevo a la cola. La función escribe los mensajes en un flujo de Registros de Amazon CloudWatch. En el siguiente diagrama, se muestran los recursos de AWS que utiliza para completar el tutorial.



Para completar este tutorial, lleve a cabo los siguientes pasos:

1. Cree una función de Lambda que escriba mensajes en Registros de CloudWatch.
2. Crear una cola de Amazon SQS.
3. Cree la asignación de orígenes de eventos de Lambda. La asignación de orígenes de eventos lee la cola de Amazon SQS e invoca la función de Lambda cuando se agrega un mensaje nuevo.
4. Para probar la configuración, agregue mensajes a la cola y supervise los resultados en Registros de CloudWatch.

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica

recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Signing in as the root user](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

### Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

## Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

## Instalar la AWS Command Line Interface

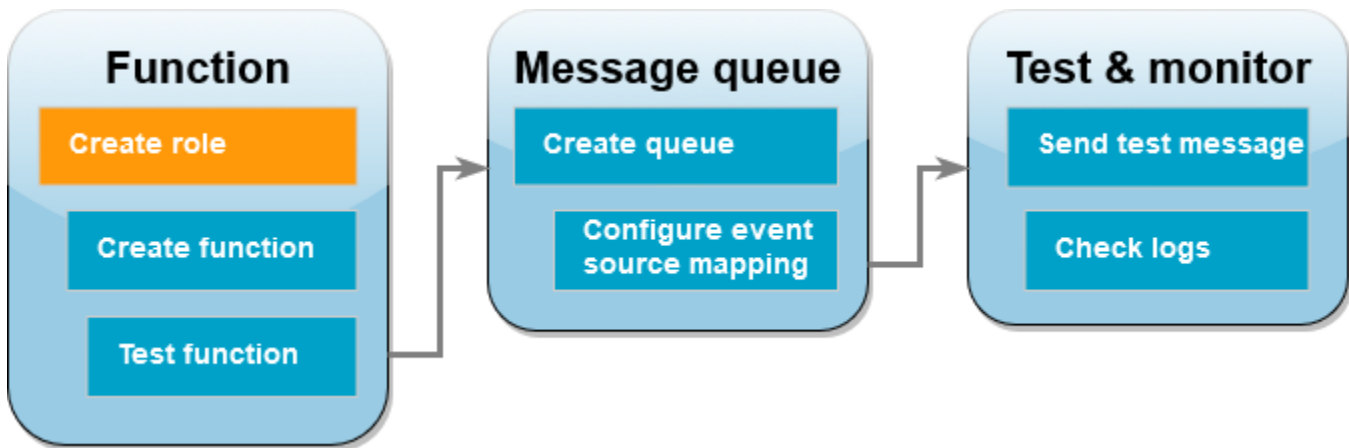
Si aún no ha instalado AWS Command Line Interface, siga los pasos que se indican en [Instalación o actualización de la versión más reciente de AWS CLI](#) para instalarlo.

El tutorial requiere un intérprete de comandos o un terminal de línea de comando para ejecutar los comandos. En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#).

## Creación del rol de ejecución



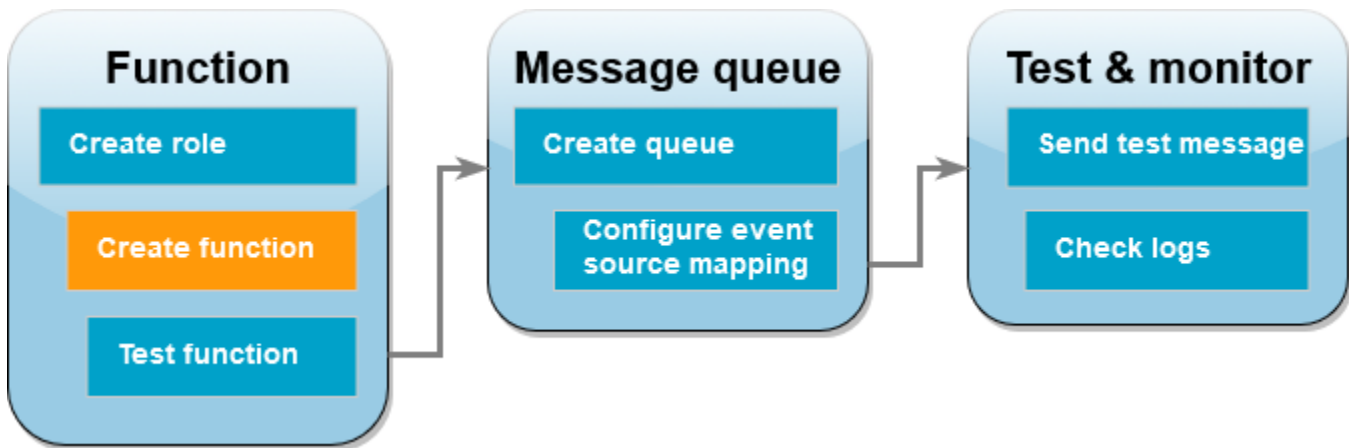
Un [rol de ejecución](#) es un rol de AWS Identity and Access Management (IAM) que concede a la función de Lambda permiso para acceder a servicios y recursos de AWS. Para permitir que la función lea elementos de Amazon SQS, adjunte la política de permisos `AWSLambdaSQSQueueExecutionRole`.

Para crear un rol de ejecución y adjuntar una política de permisos de Amazon SQS

1. Abra la página [Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. En Tipo de entidad de confianza, elija Servicio de AWS.
4. En Caso de uso, elija Lambda.
5. Elija Siguiente.
6. En el cuadro de búsqueda Permisos de selección, ingrese **`AWSLambdaSQSQueueExecutionRole`**.
7. Seleccione la política `AWSLambdaSQSQueueExecutionRole` y luego elija Siguiente.
8. En Detalles del rol, para Nombre del rol, ingrese **`lambda-sqs-role`**; luego elija Crear rol.

Tras crear la función, anote el nombre de recurso de Amazon (ARN) de la función de ejecución. Lo necesitará en pasos posteriores.

## Creación de la función



Cree una función de Lambda: que procese los mensajes de Amazon SQS. El código de función registra el cuerpo del mensaje de Amazon SQS en los Registros de CloudWatch.

En este tutorial, se utiliza el tiempo de ejecución de Node.js 18.x, pero también hemos proporcionado archivos de código de ejemplo en otros lenguajes de tiempo de ejecución. Puede seleccionar la pestaña del siguiente cuadro para ver el código del tiempo de ejecución que le interesa. El código JavaScript que usará en este paso está en el primer ejemplo que se muestra en la pestaña JavaScript.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SQS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
using Amazon.Lambda.Core;  
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }


    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            //Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```



## Go

## SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de SQS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
    for _, record := range event.Records {
        err := processMessage(record)
        if err != nil {
            return err
        }
    }
    fmt.Println("done")
    return nil
}

func processMessage(record events.SQSMessage) error {
    fmt.Printf("Processed message %s\n", record.Body)
    // TODO: Do interesting work based on the new message
    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
    }
}
```

```
}  
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
exports.handler = async (event, context) => {  
  for (const message of event.Records) {  
    await processMessageAsync(message);  
  }  
  console.info("done");  
};  
  
async function processMessageAsync(message) {  
  try {  
    console.log(`Processed message ${message.body}`);  
    // TODO: Do interesting work based on the new message  
    await Promise.resolve(1); //Placeholder for actual async work  
  } catch (err) {  
    console.error("An error occurred");  
    throw err;  
  }  
}
```

### Uso de un evento de SQS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SQS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SQS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].each do |message|
        process_message(message)
    end
    puts "done"
end

def process_message(message)
    begin
        puts "Processed message #{message['body']}"
        # TODO: Do interesting work based on the new message
    end
end
```

```
rescue StandardError => err
  puts "An error occurred"
  raise err
end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
            record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
}
```

```

    .init();

    run(service_fn(function_handler)).await
  }

```

Para crear una función de Lambda Node.js.

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```

mkdir sqs-tutorial
cd sqs-tutorial

```

2. Copie el código de muestra de JavaScript en un nuevo archivo con el nombre `index.js`.
3. Cree un paquete de implementación utilizando el siguiente comando `zip`.

```

zip function.zip index.js

```

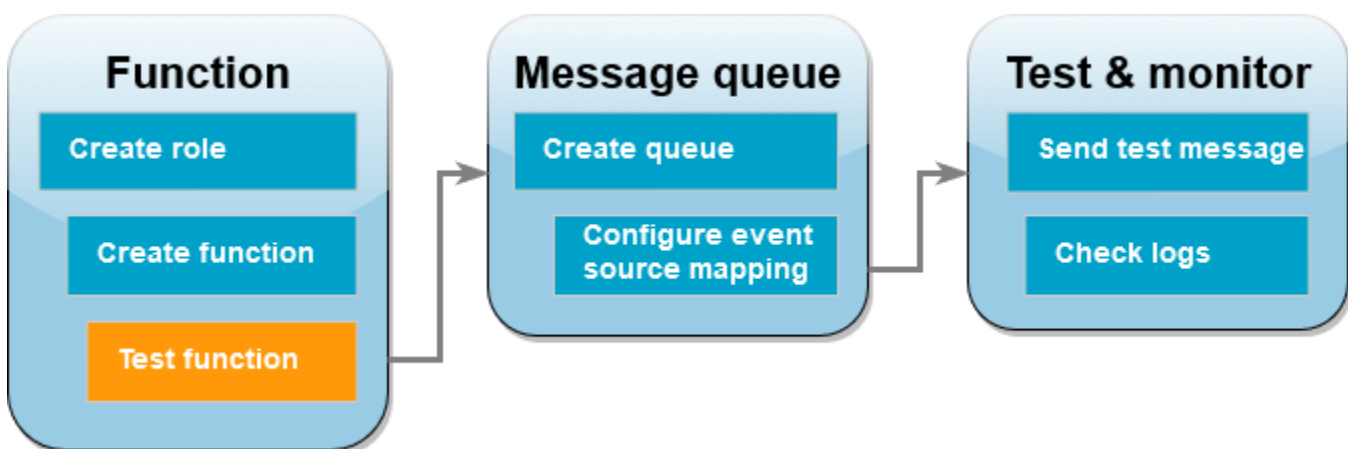
4. Cree una función de Lambda con el comando de AWS CLI [create-function](#). Para el parámetro `role`, introduzca el ARN del rol de ejecución que creó anteriormente.

```

aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::111122223333:role/lambda-sqs-role

```

Prueba de la función



Invoke la función de Lambda manualmente con el comando `invoke` AWS CLI y un evento de Amazon SQS de muestra.



## Para invocar la función de Lambda con un evento de ejemplo

1. Guarde el siguiente JSON como un archivo denominado `input.json`. Este JSON simula un evento que Amazon SQS podría enviar a la función de Lambda, donde "body" contiene el mensaje real de la cola. En este ejemplo, el mensaje es "test".

### Example Evento de Amazon SQS

Se trata de un evento de prueba y no es necesario que cambie el mensaje o el número de cuenta.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",
      "body": "test",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOL023YVJ4V0",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
      "messageAttributes": {},
      "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:my-queue",
      "awsRegion": "us-east-1"
    }
  ]
}
```

2. Ejecute el siguiente comando de AWS CLI [invoke](#). Este comando devuelve Registros de CloudWatch en la respuesta. Para obtener más información acerca de la recuperación de registros, consulte [Acceso a los registros con la AWS CLI](#).

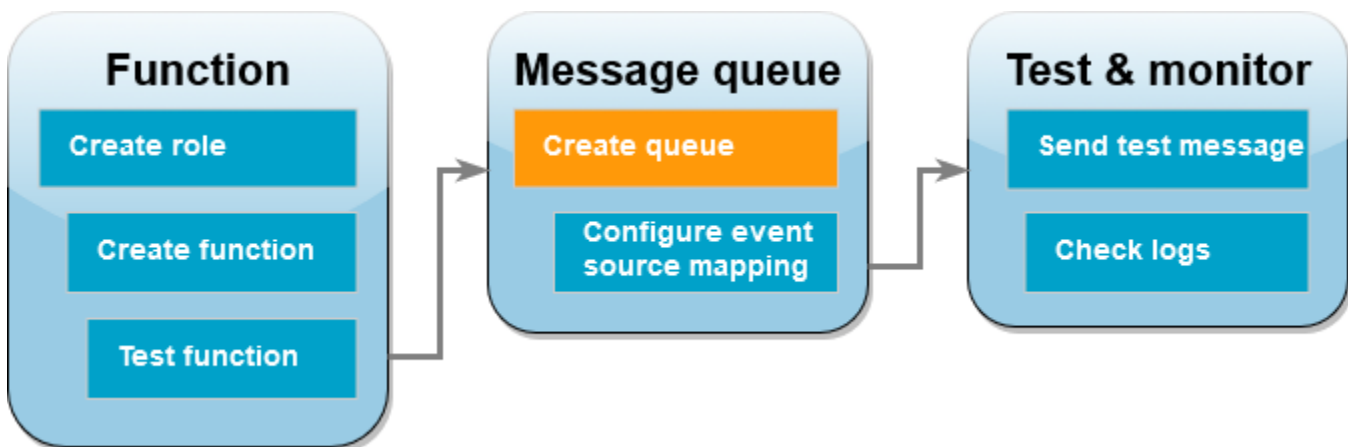
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
--log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

- Encuentre el registro INFO en la respuesta. Aquí es donde la función de Lambda registra el cuerpo del mensaje. Debería ver registros con un aspecto similar al siguiente:

```
2023-09-11T22:45:04.271Z 348529ce-2211-4222-9099-59d07d837b60 INFO Processed
message test
2023-09-11T22:45:04.288Z 348529ce-2211-4222-9099-59d07d837b60 INFO done
```

Cree una cola de Amazon SQS.



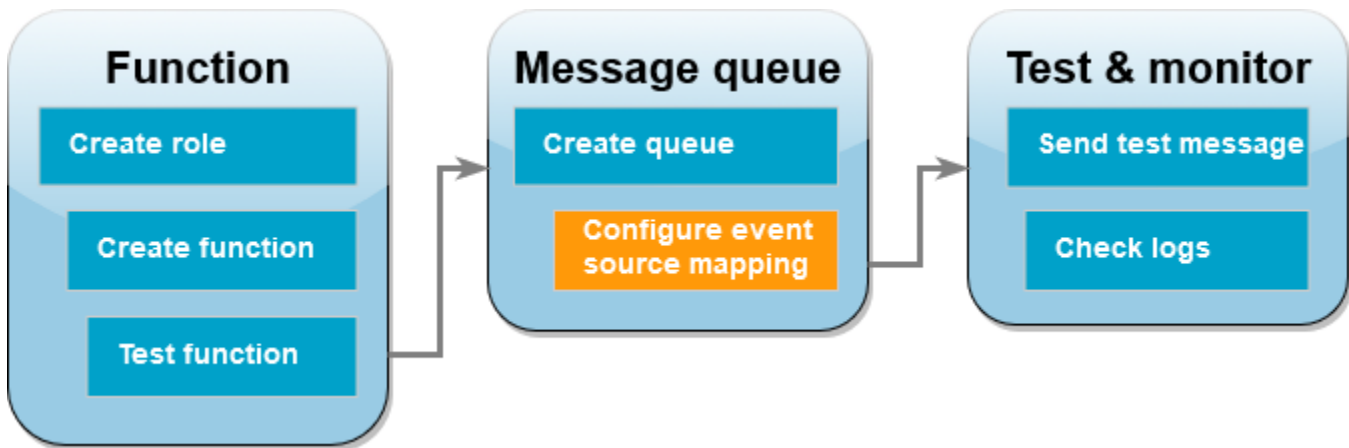
Cree una cola de Amazon SQS que la función de Lambda pueda utilizar como fuente de eventos.

Para crear una cola

- Abra la [consola de Amazon SQS](#).
- Elija Crear cola.
- Escriba un nombre para la cola. Deje las configuraciones predeterminadas de todas las demás opciones.
- Elija Crear cola.

Tras crear la cola, anote su ARN. Lo necesitará en el siguiente paso al asociar la cola a la función de Lambda.

## Configuración del origen de eventos



Conecte la cola de Amazon SQS a la función de Lambda mediante la creación de una [asignación de orígenes de eventos](#). La asignación de orígenes de eventos lee la cola de Amazon SQS e invoca la función de Lambda cuando se agrega un mensaje nuevo.

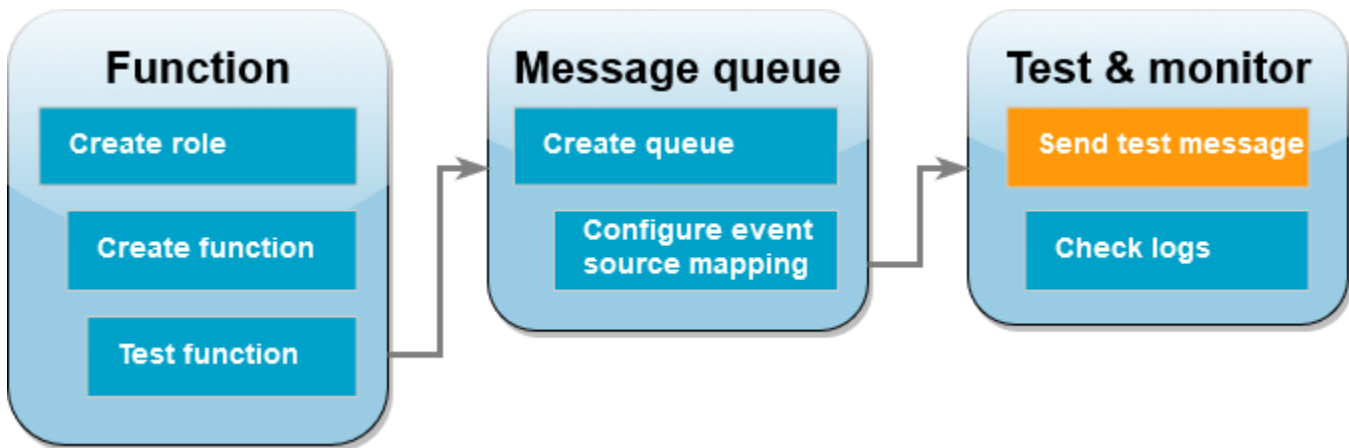
Para crear una asignación entre la cola de Amazon SQS y la función de Lambda, ejecute el comando de AWS CLI [create-event-source-mapping](#). Ejemplo:

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Para obtener una lista de las asignaciones de orígenes de eventos, utilice el comando [list-event-source-mappings](#). Ejemplo:

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

## Enviar un mensaje de prueba

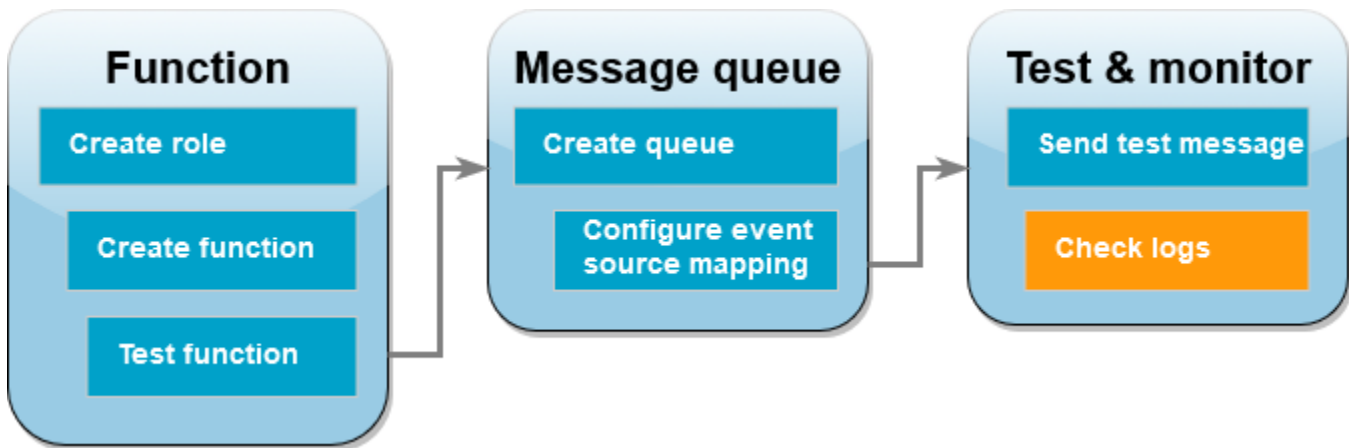


Para enviar un mensaje de Amazon SQS a la función de Lambda

1. Abra la [consola de Amazon SQS](#).
2. Elija la cola que creó anteriormente.
3. Seleccione Enviar y recibir mensajes.
4. En el Cuerpo del mensaje, ingrese un mensaje de prueba, como “este es un mensaje de prueba”.
5. Elija Enviar mensaje.

Lambda sondea la cola en busca de actualizaciones. Cuando hay un nuevo mensaje, Lambda invoca la función con estos nuevos datos de evento desde la cola. Si el controlador de la función vuelve sin excepciones, Lambda considera que el mensaje se procesó correctamente y empieza a leer nuevos mensajes en la cola. Después de procesar correctamente un mensaje, Lambda lo elimina automáticamente de la cola. Si el controlador genera una excepción, Lambda considera que el lote de mensajes no se procesó correctamente e invoca la función con el mismo lote de mensajes.

## Consulta de los Registros de CloudWatch



Para confirmar que la función procesó el mensaje

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija la función ProcessSQSRecord.
3. Elegir Monitor.
4. Seleccione Ver Registros en CloudWatch.
5. En la consola de CloudWatch, elija el Flujo de registro para la función.
6. Busque el registro INFO. Aquí es donde la función de Lambda registra el cuerpo del mensaje. Debería ver el mensaje que envió desde la cola de Amazon SQS. Ejemplo:

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed message this is a test message.
```

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.

4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete (Eliminar).

### Para eliminar la cola de Amazon SQS

1. Inicie sesión en la AWS Management Console y abra la consola de Amazon SQS en <https://console.aws.amazon.com/sqs/>.
2. Seleccione la cola que ha creado.
3. Elija Eliminar.
4. Introduzca **confirm** en el campo de entrada de texto.
5. Elija Eliminar.

## Tutorial: Uso de una cola entre cuentas de Amazon SQS como un origen de eventos

En este tutorial, creará una función de Lambda que consuma los mensajes de una cola de Amazon Simple Queue Service (Amazon SQS) en una cuenta de AWS diferente. Este tutorial incluye dos cuentas de AWS: la Cuenta A hace referencia a la cuenta que contiene la función de Lambda y la Cuenta B hace referencia a la cuenta que contiene la cola de Amazon SQS.

### Requisitos previos

En este tutorial, se presupone que tiene algunos conocimientos sobre las operaciones básicas de Lambda y la consola de Lambda. Si aún no lo ha hecho, siga las instrucciones de [Cree una función de Lambda con la consola](#) para crear su primera función de Lambda.

Para completar los siguientes pasos, necesita la [versión 2 de la AWS Command Line Interface \(AWS CLI\)](#). Los comandos y la salida esperada se enumeran en bloques separados:

```
aws --version
```

Debería ver los siguientes datos de salida:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Para comandos largos, se utiliza un carácter de escape (\) para dividir un comando en varias líneas.

En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#). Los comandos de la CLI de ejemplo de esta guía utilizan el formato Linux. Los comandos que incluyen documentos JSON en línea deben reformatearse si utiliza la CLI de Windows.

## Creación del rol de ejecución (Cuenta A)

En la Cuenta A, cree un [rol de ejecución](#) que conceda permiso a la función para acceder a los recursos necesarios de AWS.

Para crear un rol de ejecución

1. Abra la [página de Roles](#) (Roles) en la consola de AWS Identity and Access Management (IAM).
2. Elija Crear rol.
3. Cree un rol con las propiedades siguientes.
  - Trusted entity (Entidad de confianza): AWS Lambda
  - Permisos: AWSLambdaSQSQueueExecutionRole
  - Role name (Nombre de rol): **cross-account-lambda-sqs-role**

La política AWSLambdaSQSQueueExecutionRole tiene los permisos que la función necesita para leer elementos de Amazon SQS y escribir registros en Amazon CloudWatch Logs.

## Creación de la función (Cuenta A)

En la Cuenta A, cree una función de Lambda que procese los mensajes de Amazon SQS. El siguiente ejemplo de código Node.js 18 escribe cada mensaje en un registro de Registros de CloudWatch.

### Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

### Cómo crear la función

#### Note

Al seguir estos pasos, se crea una función en Node.js 18. Para otros lenguajes, los pasos son similares, pero algunos detalles son diferentes.

1. Guarde el código de ejemplo como un archivo denominado `index.mjs`.
2. Cree un paquete de implementación.

```
zip function.zip index.mjs
```

3. Cree la función mediante el comando `create-function` de la AWS Command Line Interface (AWS CLI).

```
aws lambda create-function --function-name CrossAccountSQSExample \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
--role arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role
```



## Prueba de la función (Cuenta A)

En la Cuenta A, pruebe la función de Lambda de forma manual mediante el comando `invoke` de la AWS CLI y un evento de Amazon SQS de muestra.

Si el controlador vuelve normalmente sin excepciones, Lambda considera que el mensaje se procesó de forma correcta y empieza a leer mensajes nuevos en la cola. Después de procesar correctamente un mensaje, Lambda lo elimina automáticamente de la cola. Si el controlador genera una excepción, Lambda considera que el lote de mensajes no se procesó correctamente e invoca la función con el mismo lote de mensajes.

1. Guarde el siguiente JSON como un archivo denominado `input.txt`.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgXlaS3SLy0a...",
      "body": "test",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOL023YVJ4V0",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
      "messageAttributes": {},
      "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:example-queue",
      "awsRegion": "us-east-1"
    }
  ]
}
```

El JSON anterior simula un evento que Amazon SQS podría enviar a la función de Lambda, donde `"body"` contiene el mensaje real de la cola.

2. Ejecute el siguiente comando AWS CLI de la `invoke`.

```
aws lambda invoke --function-name CrossAccountSQSExample \
  --cli-binary-format raw-in-base64-out \
  --payload file://input.txt outputfile.txt
```

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

3. Verifique la salida en el archivo `outputfile.txt`.

## Creación de una cola de Amazon SQS (Cuenta B)

En la Cuenta B, cree una cola de Amazon SQS que la función de Lambda en la Cuenta A pueda utilizar como un origen de eventos.

Para crear una cola

1. Abra la [consola de Amazon SQS](#).
2. Elija Crear cola.
3. Cree una cola con las siguientes propiedades.
  - Type (Tipo): estándar
  - Name (Nombre): `LambdaCrossAccountQueue`
  - Configuration (Configuración): conserve la configuración predeterminada.
  - Access policy (Política de acceso): elija Advanced (Avanzada). Pegue la siguiente política JSON:

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [{
    "Sid": "Queue1_AllActions",
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "arn:aws:iam::<AccountA_ID>:role/cross-account-lambda-sqs-role"
      ]
    },
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue"
  ]
}
```

```
]
}
```

Esta política concede al rol de ejecución de Lambda en la Cuenta A los permisos para consumir mensajes de esta cola de Amazon SQS.

- Después de crear la cola, registre su Nombre de recurso de Amazon (ARN). Lo necesitará en el siguiente paso al asociar la cola a la función de Lambda.

## Configuración del origen de eventos (Cuenta A)

En la Cuenta A, cree una asignación de orígenes de eventos entre la cola de Amazon SQS en la Cuenta B y la función de Lambda al ejecutar el siguiente comando `create-event-source-mapping` de la AWS CLI.

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

Para obtener una lista de asignaciones de orígenes de eventos, ejecute el siguiente comando.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:<AccountB_ID>:LambdaCrossAccountQueue
```

## Prueba de la configuración

Ahora puede probar la configuración de la siguiente manera:

- En la Cuenta B, abra la [consola de Amazon SQS](#).
- Elija `LambdaCrossAccountQueue`, que creó anteriormente.
- Seleccione `Send and receive messages` (Enviar y recibir mensajes).
- En `Message body` (Cuerpo del mensaje), ingrese un mensaje de prueba.
- Elija `Enviar mensaje`.

La función de Lambda en la Cuenta A debería recibir el mensaje. Lambda continuará sondeando la cola en busca de actualizaciones. Cuando hay un nuevo mensaje, Lambda invoca la función con estos nuevos datos de evento desde la cola. La función se ejecuta y crea registros en Amazon CloudWatch. Puede ver los registros en la [consola de CloudWatch](#).

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

En la Cuenta A, limpie el rol de ejecución y la función de Lambda.

### Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete (Eliminar).

En la Cuenta B, limpie la cola de Amazon SQS.

### Para eliminar la cola de Amazon SQS

1. Inicie sesión en la AWS Management Console y abra la consola de Amazon SQS en <https://console.aws.amazon.com/sqs/>.
2. Seleccione la cola que ha creado.
3. Elija Eliminar.
4. Introduzca **confirm** en el campo de entrada de texto.
5. Elija Eliminar.

# Invocación de una función de Lambda con eventos por lotes de Amazon S3

Puede utilizar las operaciones por lotes de Amazon S3 para invocar una función de Lambda en un conjunto grande de objetos de Amazon S3. Amazon S3 realiza un seguimiento del progreso de las operaciones por lotes, envía notificaciones y almacena un informe de finalización que muestra el estado de cada acción.

Para ejecutar una operación por lotes, debe crear un [trabajo de operaciones por lotes](#) de Amazon S3. Cuando se crea el trabajo, se proporciona un manifiesto (la lista de objetos) y se configura la acción a realizar en esos objetos.

Cuando se inicia el trabajo por lotes, Amazon S3 invoca la función de Lambda [sincrónicamente](#) para cada objeto del manifiesto. El parámetro de evento incluye los nombres del bucket y del objeto.

En el ejemplo siguiente se muestra el evento que Amazon S3 envía a la función de Lambda de un objeto denominado customerImage1.jpg en el bucket amzn-s3-demo-bucket.

Example Evento de solicitud por lotes de Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "job": {
    "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
  },
  "tasks": [
    {
      "taskId": "dGFza2lkZ291c2hlcmUK",
      "s3Key": "customerImage1.jpg",
      "s3VersionId": "1",
      "s3BucketArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
  ]
}
```

Su función de Lambda debe devolver un objeto JSON con los campos tal como se muestran en el siguiente ejemplo. Puede copiar el parámetro `invocationId` y `taskId` desde el parámetro de evento. Puede devolver una cadena en el `resultString`. Amazon S3 guarda los valores `resultString` en el informe de finalización.

## Example Respuesta de solicitud por lotes de Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

## Invocación de una función de Lambda desde operaciones por lotes de Amazon S3

Puede invocar la función de Lambda con el ARN de una función no calificada o calificada. Si desea utilizar la misma versión de una función para todo el trabajo por lotes, configure una versión de función específica en el parámetro `FunctionARN` cuando cree el trabajo. Si configura un alias o el calificador `$LATEST`, el trabajo por lotes comienza inmediatamente a llamar a la nueva versión de la función si el alias o `$LATEST` se actualizan durante la ejecución del trabajo.

Tenga en cuenta que no puede reutilizar una función basada en eventos de Amazon S3 existente para operaciones por lotes. Esto se debe a que la operación por lotes de Amazon S3 pasa un parámetro de evento diferente a la función de Lambda y espera un mensaje de retorno con una estructura JSON específica.

En la [política basada en recursos](#) que cree para el trabajo por lotes de Amazon S3, asegúrese de establecer permisos para que el trabajo invoque su función de Lambda.

En el [rol de ejecución](#) de la función, establezca una política de confianza para que Amazon S3 asuma el rol cuando ejecute la función.

Si su función utiliza SDK de AWS para administrar recursos de Amazon S3, debe agregar permisos de Amazon S3 a la función de ejecución.

Cuando se ejecuta el trabajo, Amazon S3 inicia varias instancias de función para procesar los objetos de Amazon S3 en paralelo, hasta el [límite de simultaneidad](#) de la función. Amazon S3 limita el aumento inicial de instancias para evitar un costo excesivo para trabajos más pequeños.

Si la función de Lambda devuelve un código de respuesta `TemporaryFailure`, Amazon S3 reintenta la operación.

Para obtener más información acerca de las operaciones por lotes de Amazon S3, consulte [Administrar operaciones por lotes](#) en la Guía para desarrolladores de Amazon S3.

Para obtener un ejemplo de cómo utilizar una función Lambda en operaciones por lotes de Amazon S3, consulte [Invocación de una función de Lambda desde las operaciones por lotes de Amazon S3](#) en la Guía para desarrolladores de Amazon S3.

# Invocar las funciones de Lambda usando las notificaciones de Amazon SNS

Puede utilizar una función de Lambda para procesar notificaciones de Amazon Simple Notification Service (Amazon SNS). Amazon SNS admite funciones de Lambda como destino para los mensajes enviados a un tema. Puede suscribir la función a temas de la misma cuenta o de otras cuentas de AWS. Para ver un tutorial detallado, consulte [the section called “Tutorial”](#).

Lambda admite desencadenadores de SNS únicamente para temas de SNS estándar. Los temas FIFO no son compatibles.

Para la invocación asíncrona, Lambda pone en cola el mensaje y administra los reintentos. Si Amazon SNS no puede conectar con Lambda o se rechaza el mensaje, Amazon SNS lo vuelve a intentar a intervalos cada vez mayores durante varias horas. Para conocer los detalles, consulte [Fiabilidad](#) en las preguntas frecuentes de Amazon SNS.

## Warning

Las asignaciones de orígenes de eventos de Lambda procesan cada evento al menos una vez, y puede producirse un procesamiento duplicado de registros. Para evitar posibles problemas relacionados con la duplicación de eventos, le recomendamos encarecidamente que haga que el código de la función sea idempotente. Para obtener más información, consulte [¿Cómo puedo hacer que mi función de Lambda sea idempotente?](#) en el Centro de conocimientos de AWS.

## Temas

- [Adición de un desencadenador de temas de Amazon SNS para una función de Lambda mediante la consola](#)
- [Adición manual de un desencadenador de temas de Amazon SNS para una función de Lambda](#)
- [Ejemplo de forma de evento SNS](#)
- [Tutorial: Uso de AWS Lambda con Amazon Simple Notification Service](#)



## Adición de un desencadenador de temas de Amazon SNS para una función de Lambda mediante la consola

Para agregar un tema de SNS como desencadenador de una función de Lambda, la forma más sencilla es utilizar la consola de Lambda. Al agregar el desencadenador a través de la consola, Lambda configura automáticamente los permisos y suscripciones necesarios para empezar a recibir eventos del tema de SNS.

Cómo agregar un tema de SNS como desencadenador de una función de Lambda (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función a la que desee agregarle el desencadenador.
3. Elija Configuración y, a continuación, seleccione Desencadenadores.
4. Elija Add trigger (Añadir disparador).
5. En Configuración del desencadenador, seleccione SNS de la lista desplegable.
6. Para el tema de SNS, elija el tema de SNS al que desee suscribirse.

## Adición manual de un desencadenador de temas de Amazon SNS para una función de Lambda

Para configurar manualmente un desencadenador de SNS para una función de Lambda, debe completar los siguientes pasos:

- Defina una política basada en recursos para la función de modo que SNS pueda invocarla.
- Suscriba la función de Lambda al tema de Amazon SNS.

### Note

Si su tema de SNS y su función de Lambda están en cuentas de AWS diferentes, también debe conceder permisos adicionales para permitir las suscripciones multicuenta al tema de SNS. Para obtener más información, consulte [Conceder permiso entre cuentas para la suscripción a Amazon SNS](#).

Puede usar el AWS Command Line Interface (AWS CLI) para completar estos dos pasos. En primer lugar, para definir una política basada en recursos para una función de Lambda que permita las

invocaciones de SNS, utilice el siguiente comando de la AWS CLI. Asegúrese de sustituir el valor de `--function-name` por el nombre de la función de Lambda y el valor de `--source-arn` por el ARN del tema de SNS.

```
aws lambda add-permission --function-name example-function \  
  --source-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \  
  --statement-id function-with-sns --action "lambda:InvokeFunction" \  
  --principal sns.amazonaws.com
```

Utilice el siguiente comando AWS CLI para suscribir la función al tema SNS. Sustituya el valor de `--topic-arn` por el ARN del tema de SNS y el valor de `--notification-endpoint` por el ARN de la función de Lambda.

```
aws sns subscribe --protocol lambda \  
  --region us-east-1 \  
  --topic-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \  
  --notification-endpoint arn:aws:lambda:us-east-1:123456789012:function:example-  
function
```

## Ejemplo de forma de evento SNS

Amazon SNS invoca la función [de forma asíncrona](#) con un evento que contiene un mensaje y metadatos.

### Example Evento de mensajes de Amazon SNS

```
{  
  "Records": [  
    {  
      "EventVersion": "1.0",  
      "EventSubscriptionArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda:21be56ed-  
a058-49f5-8c98-aedd2564c486",  
      "EventSource": "aws:sns",  
      "Sns": {  
        "SignatureVersion": "1",  
        "Timestamp": "2019-01-02T12:45:07.000Z",  
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",  
        "SigningCertURL": "https://sns.us-east-1.amazonaws.com/  
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",  
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
        "Message": "Hello from SNS!",
```

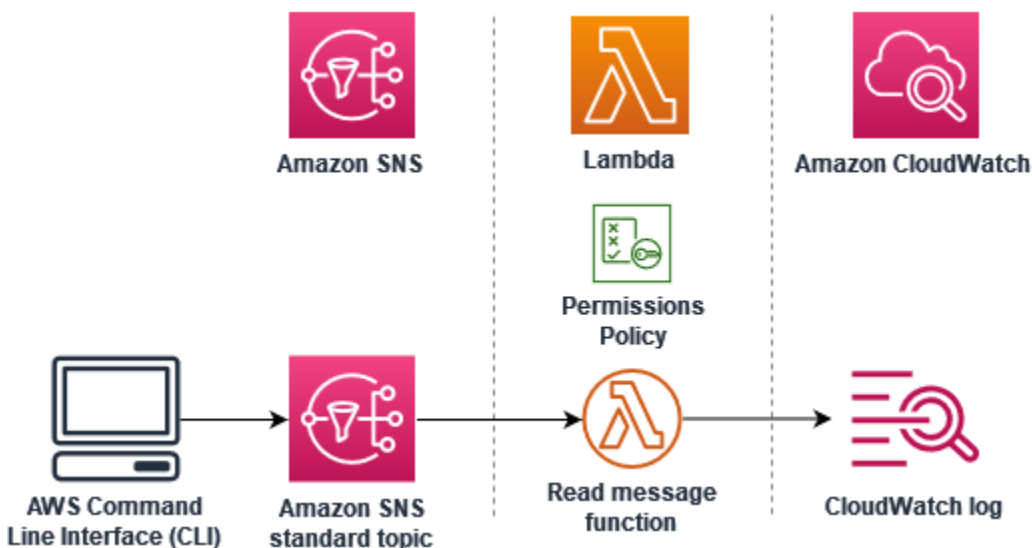
```

    "MessageAttributes": {
      "Test": {
        "Type": "String",
        "Value": "TestString"
      },
      "TestBinary": {
        "Type": "Binary",
        "Value": "TestBinary"
      }
    },
    "Type": "Notification",
    "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
    "TopicArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda",
    "Subject": "TestInvoke"
  }
}
]
}

```

## Tutorial: Uso de AWS Lambda con Amazon Simple Notification Service

En este tutorial, utilizará una función de Lambda en una Cuenta de AWS para suscribirse a un tema de Amazon Simple Notification Service (Amazon SNS) en una Cuenta de AWS independiente. Cuando publica mensajes en su tema de Amazon SNS, la función de Lambda lee el contenido del mensaje y lo envía a Registros de Amazon CloudWatch. Para completar este tutorial, debe usar la AWS Command Line Interface (AWS CLI).



Para completar este tutorial, lleve a cabo los siguientes pasos:

- En la cuenta A, cree un tema de Amazon SNS.
- En la cuenta B, cree una función de Lambda que lea los mensajes del tema.
- En la cuenta B, cree una suscripción al tema.
- Publique los mensajes en el tema de Amazon SNS en la cuenta A y confirme que la función de Lambda de la cuenta B los envíe a Registros de CloudWatch.

Al completar estos pasos, aprenderá a configurar un tema de Amazon SNS para invocar una función de Lambda. También aprenderá a crear una política de AWS Identity and Access Management (IAM) que conceda permiso a un recurso de otra Cuenta de AWS para invocar Lambda.

En el tutorial, utiliza dos Cuentas de AWS independientes. Los comandos de AWS CLI ilustran esto mediante dos perfiles con nombre llamados `accountA` y `accountB`, cada uno configurado para usarse con una Cuenta de AWS diferente. A fin de aprender a configurar AWS CLI para usar diferentes perfiles, consulte [Opciones de los archivos de configuración y credenciales](#) en la Guía del usuario de la versión 2 de la AWS Command Line Interface. Asegúrese de configurar la misma Región de AWS predeterminada para ambos perfiles.

Si los perfiles de AWS CLI que crea para las dos Cuentas de AWS utilizan nombres distintos, o si usa el perfil predeterminado y un perfil con nombre, modifique los comandos de AWS CLI en los siguientes pasos según sea necesario.

## Requisitos previos

### Registro en una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

### Procedimiento para registrarse en Cuenta de AWS

1. Abra <https://portal.aws.amazon.com/billing/signup>.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica

recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar [tareas que requieren acceso de usuario raíz](#).

AWS le enviará un email de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrar la cuenta en cualquier momento entrando en <https://aws.amazon.com/> y seleccionando Mi cuenta.

### Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no utilizar el usuario raíz en las tareas cotidianas.

### Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en [AWS Management Console](#) como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

Para obtener ayuda para iniciar sesión con el usuario raíz, consulte [Iniciar sesión como usuario raíz](#) en la Guía del usuario de AWS Sign-In.

2. Active la autenticación multifactor (MFA) para el usuario raíz.

Para obtener instrucciones, consulte [Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS \(consola\)](#) en la Guía del usuario de IAM.

### Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en [Activar AWS IAM Identity Center](#) en la Guía del usuario de AWS IAM Identity Center.

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

Para ver un tutorial sobre cómo utilizar Directorio de IAM Identity Center como origen de identidad, consulte [Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado](#) en la Guía del usuario de AWS IAM Identity Center.

## Iniciar sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, utilice la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario del IAM Identity Center, consulte [Inicio de sesión en el portal de acceso de AWS](#) en la Guía del usuario de AWS Sign-In.

## Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

Para conocer las instrucciones, consulte [Create a permission set](#) en la Guía del usuario de AWS IAM Identity Center.

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

Para conocer las instrucciones, consulte [Add groups](#) en la Guía del usuario de AWS IAM Identity Center.

## Instalar la AWS Command Line Interface

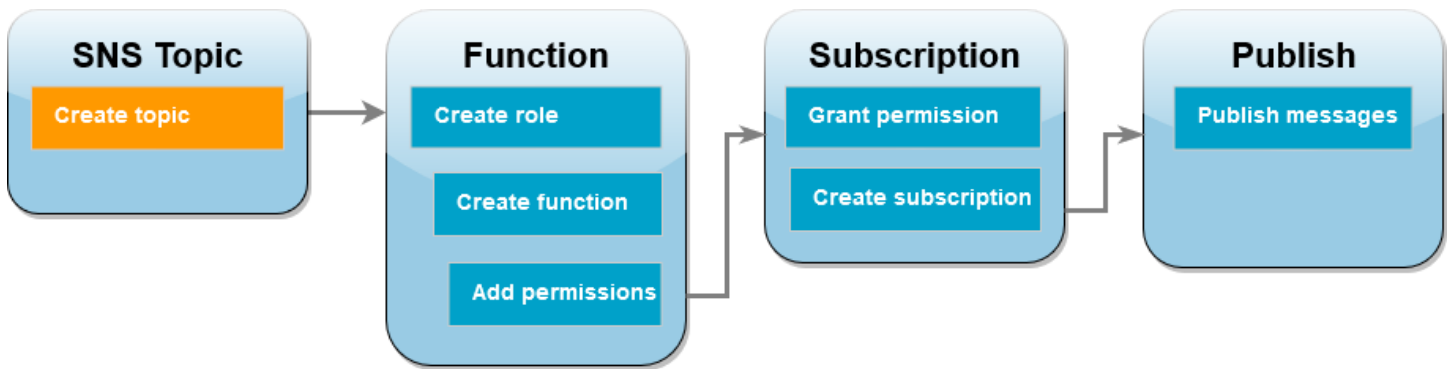
Si aún no ha instalado AWS Command Line Interface, siga los pasos que se indican en [Instalación o actualización de la versión más reciente de AWS CLI](#) para instalarlo.

El tutorial requiere un intérprete de comandos o un terminal de línea de comando para ejecutar los comandos. En Linux y macOS, use su administrador de intérprete de comandos y paquetes preferido.

### Note

En Windows, algunos comandos de la CLI de Bash que se utilizan habitualmente con Lambda (por ejemplo, zip) no son compatibles con los terminales integrados del sistema operativo. Para obtener una versión de Ubuntu y Bash integrada con Windows, [instale el subsistema de Windows para Linux](#).

## Creación de un tema de Amazon SNS (cuenta A)



Para crear el tema

- En la cuenta A, cree un tema estándar de Amazon SNS mediante el siguiente comando de AWS CLI.

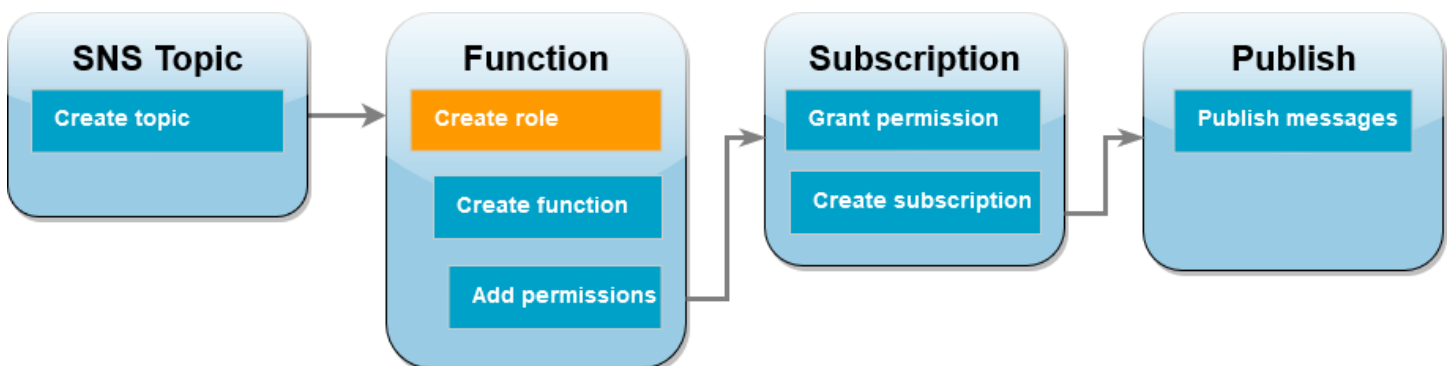
```
aws sns create-topic --name sns-topic-for-lambda --profile accountA
```

Debería ver un resultado similar a este.

```
{
  "TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
}
```

Anote el Nombre de recurso de Amazon (ARN) del tema. Lo necesitará más adelante en el tutorial cuando agregue permisos a la función de Lambda para suscribirse al tema.

## Creación de un rol de ejecución de función (cuenta B)

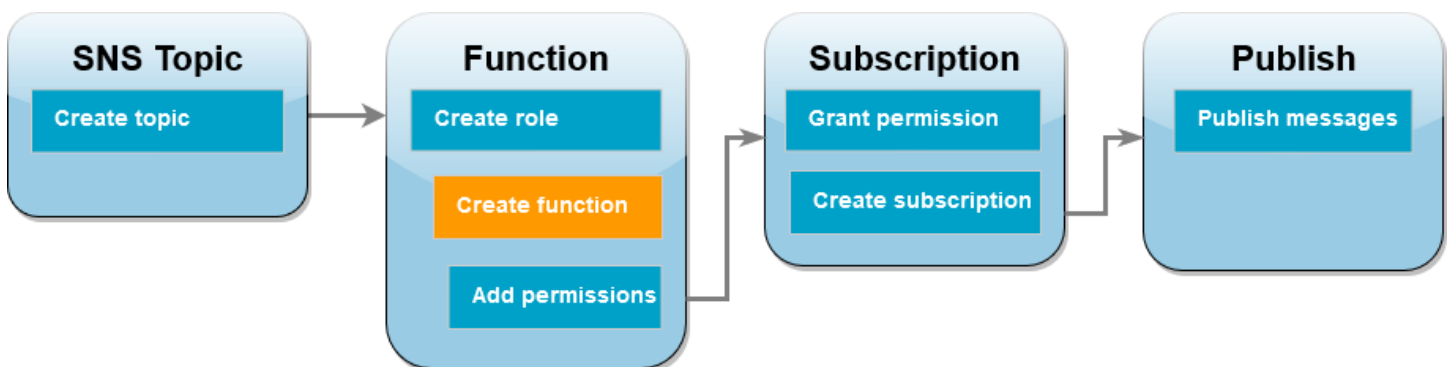


Un rol de ejecución es un rol de IAM que concede a la función de Lambda permiso para acceder a servicios y recursos de AWS. Antes de crear la función en la cuenta B, cree un rol que conceda a la función permisos básicos para escribir registros en Registros de CloudWatch. Agregaremos los permisos para leer el tema de Amazon SNS en un paso posterior.

Para crear un rol de ejecución

1. En la cuenta B, abra la [página de roles](#) en la consola de IAM.
2. Elija Crear rol.
3. En Tipo de entidad de confianza, elija Servicio de AWS.
4. En Caso de uso, elija Lambda.
5. Elija Siguiente.
6. Siga estos pasos para agregar una política de permisos básicos al rol:
  - a. En el cuadro de búsqueda Permisos de selección, ingrese **AWSLambdaBasicExecutionRole**.
  - b. Elija Siguiente.
7. Siga estos pasos para finalizar la creación del rol:
  - a. En Detalles del rol, escriba **lambda-sns-role** en Nombre de rol.
  - b. Elija Crear rol.

Creación de una función de Lambda (cuenta B)



Cree una función de Lambda que procese los mensajes de Amazon SNS. El código de función registra el contenido de los mensajes de cada entrada en los Registros de Amazon CloudWatch.

En este tutorial, se utiliza el tiempo de ejecución de Node.js 18.x, pero también hemos proporcionado archivos de código de ejemplo en otros lenguajes de tiempo de ejecución. Puede seleccionar



la pestaña del siguiente cuadro para ver el código del tiempo de ejecución que le interesa. El código JavaScript que usará en este paso está en el primer ejemplo que se muestra en la pestaña JavaScript.

## .NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
        ILambdaContext context)
```

```
{
    try
    {
        context.Logger.LogInformation($"Processed record
{record.Sns.Message}");

        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

## Go

### SDK para Go V2

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)
```

```
func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        processMessage(record)
    }
    fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
    message := record.SNS.Message
    fmt.Printf("Processed message: %s\n", message)
    // TODO: Process your record here
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;
```

```
import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

### Uso de un evento de SNS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
    ): Promise<void> => {
      for (const record of event.Records) {
        await processMessageAsync(record);
      }
      console.info("done");
    };

    async function processMessageAsync(record: SNSEventRecord): Promise<any> {
      try {
        const message: string = JSON.stringify(record.Sns.Message);
        console.log(`Processed message ${message}`);
        await Promise.resolve(1); //Placeholder for actual async work
      } catch (err) {
        console.error("An error occurred");
        throw err;
      }
    }
  }
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
```

```
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/  
*/  
  
// Additional composer packages may be required when using Bref or any other PHP  
// functions runtime.  
// require __DIR__ . '/vendor/autoload.php';  
  
use Bref\Context\Context;  
use Bref\Event\Sns\SnsEvent;  
use Bref\Event\Sns\SnsHandler;  
  
class Handler extends SnsHandler  
{  
    public function handleSns(SnsEvent $event, Context $context): void  
    {  
        foreach ($event->getRecords() as $record) {  
            $message = $record->getMessage();  
  
            // TODO: Implement your custom processing logic here  
            // Any exception thrown will be logged and the invocation will be  
            marked as failed  
  
            echo "Processed Message: $message" . PHP_EOL;  
        }  
    }  
}  
  
return new Handler();
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].map { |record| process_message(record) }
end

def process_message(record)
    message = record['Sns']['Message']
    puts("Processing message: #{message}")
rescue StandardError => e
    puts("Error processing message: #{e}")
```



```
    raise
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features
//   = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features =
//   ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
}
```

```

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```

## Cómo crear la función

1. Cree un directorio para el proyecto y, a continuación, cambie a ese directorio.

```

mkdir sns-tutorial
cd sns-tutorial

```

2. Copie el código de muestra de JavaScript en un nuevo archivo con el nombre `index.js`.
3. Cree un paquete de implementación utilizando el siguiente comando `zip`.

```

zip function.zip index.js

```

4. Ejecute el siguiente comando de AWS CLI para crear la función de Lambda en la cuenta B.

```

aws lambda create-function --function-name Function-With-SNS \
    --zip-file fileb://function.zip --handler index.handler --runtime nodejs18.x \
    --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role \
    --timeout 60 --profile accountB

```

Debería ver un resultado similar a este.

```

{
  "FunctionName": "Function-With-SNS",

```

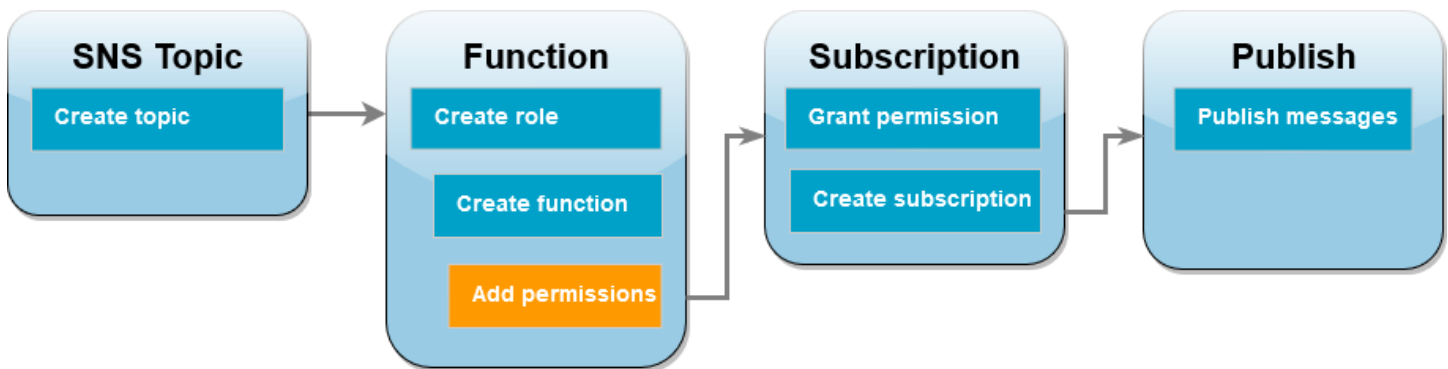
```

"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:Function-With-
SNS",
"Runtime": "nodejs18.x",
"Role": "arn:aws:iam::123456789012:role/lambda_basic_role",
"Handler": "index.handler",
...
"RuntimeVersionConfig": {
  "RuntimeVersionArn": "arn:aws:lambda:us-
west-2::runtime:7d5f06b69c951da8a48b926ce280a9daf2e8bb1a74fc4a2672580c787d608206"
}
}

```

5. Registre el Nombre de recurso de Amazon (ARN) de la función. Lo necesitará más adelante en el tutorial cuando agregue permisos para permitir a Amazon SNS que invoque la función.

### Adición de permisos a la función (cuenta B)



Para que Amazon SNS invoque la función, debe concederle permiso en una instrucción de una [política basada en recursos](#). Para agregar esta instrucción, utilice el comando `add-permission` de AWS CLI.

Para conceder permiso a Amazon SNS para invocar la función

- En la cuenta B, ejecute el siguiente comando de AWS CLI mediante el ARN del tema de Amazon SNS que registró anteriormente.

```

aws lambda add-permission --function-name Function-With-SNS \
  --source-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --statement-id function-with-sns --action "lambda:InvokeFunction" \
  --principal sns.amazonaws.com --profile accountB

```

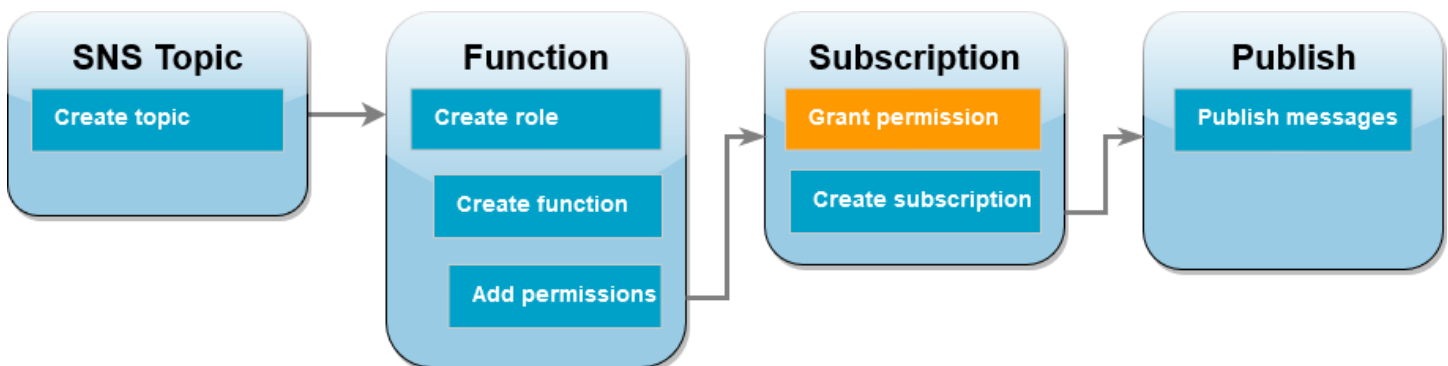
Debería ver un resultado similar a este.

```
{
  "Statement": "{ \"Condition\": { \"ArnLike\": { \"AWS:SourceArn\": \"arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda\" } }, \"Action\": [ \"lambda:InvokeFunction\" ], \"Resource\": \"arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-SNS\" , \"Effect\": \"Allow\", \"Principal\": { \"Service\": \"sns.amazonaws.com\" }, \"Sid\": \"function-with-sns\" }"
}
```

### Note

Si la cuenta con el tema de Amazon SNS está alojada en una [Región de AWS con suscripción](#), debe especificar la región en la entidad principal. Por ejemplo, si trabaja con un tema de Amazon SNS en la región de Asia-Pacífico (Hong Kong), debe especificar `sns.ap-east-1.amazonaws.com` en lugar de `sns.amazonaws.com` para la entidad principal.

## Concesión de permiso entre cuentas para la suscripción a Amazon SNS (cuenta A)



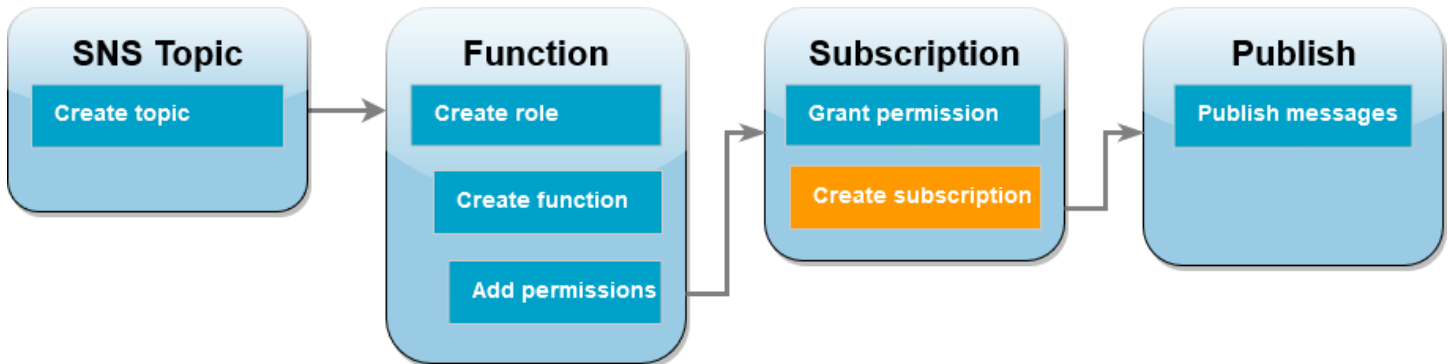
Para que su función de Lambda de la cuenta B se suscriba al tema de Amazon SNS que creó en la cuenta A, tiene que conceder permiso a la cuenta B para suscribirse al tema. Utilice el comando `add-permission` de AWS CLI para conceder este permiso.

Para conceder permiso para que la cuenta B se suscriba al tema

- En la cuenta A, ejecute el siguiente comando de AWS CLI. Utilice el ARN del tema de Amazon SNS que registró anteriormente.

```
aws sns add-permission --label lambda-access --aws-account-id <AccountB_ID> \
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --action-name Subscribe ListSubscriptionsByTopic --profile accountA
```

## Creación de una suscripción (cuenta B)



En la cuenta B, ahora puede suscribir su función de Lambda al tema de Amazon SNS que creó al principio del tutorial en la cuenta A. Cuando se envía un mensaje a este tema (`sns-topic-for-lambda`), Amazon SNS invoca su función de Lambda `Function-With-SNS` en la cuenta B.

Para crear una suscripción

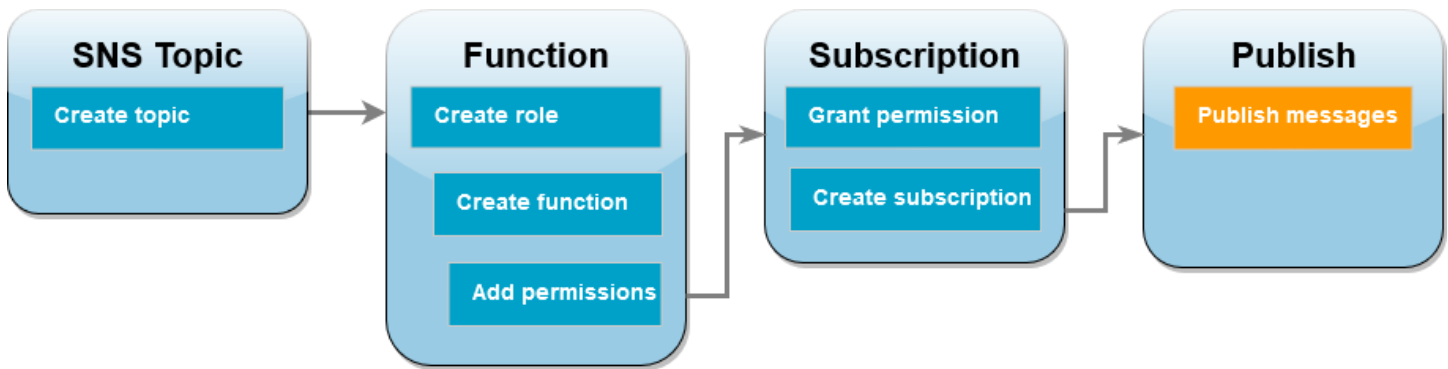
- En la cuenta B, ejecute el siguiente comando de AWS CLI. Use la región predeterminada en la que creó el tema y los ARN del tema y la función de Lambda.

```
aws sns subscribe --protocol lambda \
  --region us-east-1 \
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --notification-endpoint arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-SNS \
  --profile accountB
```

Debería ver un resultado similar a este.

```
{
  "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

## Publicación de mensajes en el tema (cuenta A y cuenta B)



Ahora que su función de Lambda de la cuenta B se ha suscrito a su tema de Amazon SNS en la cuenta A, es hora de probar la configuración mediante la publicación de mensajes en el tema. Para confirmar que Amazon SNS ha invocado la función de Lambda, utilice Registros de CloudWatch para ver la salida de la función.

Para publicar un mensaje en el tema y ver la salida de la función

1. Escriba `Hello World` en un archivo de texto y guárdelo como `message.txt`.
2. Desde el mismo directorio en el que guardó el archivo de texto, ejecute el siguiente comando de AWS CLI en la cuenta A. Use el ARN de su propio tema.

```
aws sns publish --message file://message.txt --subject Test \
  --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
  --profile accountA
```

Esto devolverá un ID de mensaje con un identificador único que indica que Amazon SNS ha aceptado el mensaje. Posteriormente, Amazon SNS intentará entregar el mensaje a los suscriptores del tema. Para confirmar que Amazon SNS ha invocado la función de Lambda, utilice Registros de CloudWatch para ver la salida de la función:

3. En la cuenta B, abra la página [Grupos de registro](#) de la consola de Amazon CloudWatch.
4. Elija el grupo de registro para la función (`/aws/lambda/Function-With-SNS`).
5. Elija el flujo de registros más reciente.
6. Si la función se ha invocado correctamente, verá una salida similar a la siguiente en la que se muestra el contenido del mensaje que publicó en el tema.

```
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO Processed
message Hello World
```

```
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO done
```

## Eliminación de sus recursos

A menos que desee conservar los recursos que creó para este tutorial, puede eliminarlos ahora. Si elimina los recursos de AWS que ya no utiliza, evitará gastos innecesarios en su Cuenta de AWS.

En la cuenta A, limpie el tema de Amazon SNS.

Para eliminar el tema de Amazon SNS

1. Abra la página [Topics \(Temas\)](#) en la consola de Amazon SNS.
2. Seleccione el tema que creó.
3. Elija Eliminar.
4. Introduzca **delete me** en el campo de entrada de texto.
5. Elija Eliminar.

En la cuenta B, limpie el rol de ejecución, la función de Lambda y la suscripción de Amazon SNS.

Cómo eliminar el rol de ejecución

1. Abra la página [Roles](#) en la consola de IAM.
2. Seleccione el rol de ejecución que creó.
3. Elija Eliminar.
4. Si desea continuar, escriba el nombre del rol en el campo de entrada de texto y elija Delete (Eliminar).

Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y elija Delete (Eliminar).

## Para eliminar la suscripción a Amazon SNS

1. Abra la página [Suscriptions \(Suscripciones\)](#) en la consola de Amazon SNS.
2. Seleccione la suscripción que creó.
3. Elija Delete (Eliminar), Delete (Eliminar).



# Administrar permisos en AWS Lambda

Puede utilizar AWS Identity and Access Management (IAM) para administrar permisos en AWS Lambda. Hay dos categorías principales de permisos que debe tener en cuenta al trabajar con funciones de Lambda:

- Permisos que las funciones de Lambda necesitan para realizar las acciones de la API y acceder a otros recursos de AWS.
- Permisos que otros usuarios y entidades de AWS necesitan para acceder a las funciones de Lambda

Las funciones de Lambda a menudo necesitan acceder a otros recursos de AWS y realizar diversas operaciones de la API en esos recursos. Por ejemplo, puede tener una función de Lambda que responda a un evento actualizando las entradas de una base de datos de Amazon DynamoDB. En este caso, la función necesita permisos para acceder a la base de datos, así como permisos para colocar o actualizar elementos en esa base de datos.

Usted define los permisos que necesita la función de Lambda se definen en un rol de IAM especial llamado [rol de ejecución](#). En este rol, puede adjuntar una política que defina los permisos que la función necesita para acceder a otros recursos de AWS y leer desde orígenes de eventos. Cada función de Lambda debe tener un rol de ejecución. Como mínimo, su rol de ejecución debe tener acceso a Amazon CloudWatch, ya que las funciones de Lambda se registran en Registros de CloudWatch de forma predeterminada. Puede adjuntar la [política administrada AWSLambdaBasicExecutionRole](#) a su rol de ejecución para cumplir con este requisito.

Para conceder permisos para que otras Cuentas de AWS, organizaciones y servicios accedan a sus recursos de Lambda, tiene varias opciones:

- Puede utilizar [políticas basadas en identidades](#) para conceder acceso a otros usuarios a sus recursos de Lambda. Las políticas basadas en identidad se pueden aplicar a los usuarios directamente, o a los grupos y roles que están asociados a un usuario.
- Puede usar las [políticas basadas en recursos](#) para darles permiso a otras cuentas y servicios de AWS para acceder a los recursos de Lambda. Cuando un usuario intenta acceder a un recurso de Lambda, este servicio tiene en cuenta tanto las políticas basadas en identidades como la política basada en recursos del recurso. Cuando un servicio de AWS, como Amazon Simple Storage Service (Amazon S3), llama a la función de Lambda, el servicio considera solo la política basada en recursos.

- Puede utilizar un modelo de [control de acceso basado en atributos \(ABAC\)](#) para controlar el acceso a las funciones de Lambda. Con ABAC, puede adjuntar etiquetas a una función de Lambda, pasarlas en determinadas solicitudes de la API o adjuntarlas a la entidad principal de IAM que realiza la solicitud. Especifique las mismas etiquetas en el elemento de condición de una política de IAM para controlar el acceso a la función.

EnAWS, se recomienda conceder solo los permisos necesarios para realizar una tarea (permisos de [privilegios mínimos](#)). Para implementar esto en Lambda, recomendamos comenzar con una política [administrada de AWS](#). Puede utilizar estas políticas administradas tal como están, o como punto de partida para escribir sus propias políticas más restrictivas.

Para ayudarlo a ajustar sus permisos para el acceso con privilegios mínimos, Lambda proporciona algunas condiciones adicionales que puede incluir en sus políticas. Para obtener más información, consulte [the section called “Recursos y condiciones”](#).

Para obtener más información acerca de IAM, consulte la [guía del usuario de IAM](#).

# Definición de permisos de funciones de Lambda con un rol de ejecución

Un rol de ejecución de una función de Lambda es un rol de AWS Identity and Access Management (IAM) que concede a la función permiso para acceder a servicios y recursos de AWS. Por ejemplo, puede crear un rol de ejecución que tenga permiso para enviar registros a Amazon CloudWatch y cargar datos de seguimiento en AWS X-Ray. Esta página proporciona información sobre cómo crear, ver y administrar el rol de ejecución de una función de Lambda.

Lambda asume automáticamente su rol de ejecución cuando invoca su función. Debería evitar llamar de forma manual a `sts:AssumeRole` para que asuma el rol de ejecución en el código de función. Si su caso de uso requiere que el rol se asuma por sí mismo, debe incluir al rol como una entidad principal de confianza en la política de confianza de su rol. Para obtener más información acerca de cómo modificar una política de confianza de roles, consulte [Modificación de una política de confianza de rol \(consola\)](#) en la Guía del usuario de IAM.

Para que Lambda asuma correctamente su rol de ejecución, la [política de confianza](#) del rol debe especificar la entidad principal de servicio de Lambda (`lambda.amazonaws.com`) como un servicio de confianza.

## Temas

- [Creación de un rol de ejecución en la consola de IAM](#)
- [Creación y administración de roles con la AWS CLI](#)
- [Otorgue privilegios de acceso mínimos a su rol de ejecución de Lambda](#)
- [Visualización y actualización de permisos en el rol de ejecución](#)
- [Trabajo con políticas administradas de AWS en el rol de ejecución](#)
- [Uso del ARN de la función de origen para controlar el comportamiento de acceso a la función](#)

## Creación de un rol de ejecución en la consola de IAM

De forma predeterminada, Lambda crea un rol de ejecución con permisos mínimos al [crear una función en la consola de Lambda](#). En concreto, este rol de ejecución incluye la [política administrada `AWSLambdaBasicExecutionRole`](#), que otorga a su función permisos básicos para registrar eventos de registro en Registros de Amazon CloudWatch.

Por lo general, sus funciones necesitan permisos adicionales para realizar tareas más significativas. Por ejemplo, puede tener una función de Lambda que responda a un evento actualizando las entradas de una base de datos de Amazon DynamoDB. Puede crear un rol de ejecución con los permisos necesarios mediante la consola IAM.

Para crear un rol de ejecución en la consola de IAM

1. Abra la [página Roles](#) en la consola de IAM.
2. Elija Crear rol.
3. En Tipo de entidad de confianza, seleccione Servicio de AWS.
4. En Use case (Caso de uso), elija Lambda.
5. Elija Siguiente.
6. Seleccione las políticas administradas de AWS que desee adjuntar a su rol. Por ejemplo, si la función necesita acceder a DynamoDB, seleccione la política administrada `AWSLambdaDynamoDBExecutionRole`.
7. Elija Siguiente.
8. Introduzca un nombre de rol en el campo Role name y, a continuación, elija Create role.

Para obtener instrucciones, consulte [Creating a role for an AWS service \(console\) \(Creación de un rol para un servicio de \[consola\] de AWS\)](#) en la Guía del usuario de IAM.

Después de crear su rol de ejecución, adjúntelo a su función. Cuando [crea una función en la consola de Lambda](#), puede adjuntar a la función cualquier rol de ejecución que haya creado previamente. Si desea adjuntar un nuevo rol de ejecución a una función existente, siga los pasos que se indican en .

## Creación y administración de roles con la AWS CLI

Para crear un rol de ejecución con AWS Command Line Interface (AWS CLI), utilice el comando `create-role`. Al usar este comando, puede especificar las [políticas de confianza](#) en línea. La política de confianza de un rol otorga a la entidad principal especificada permiso para asumir el rol. En el siguiente ejemplo, usted concede a la entidad principal del servicio Lambda el permiso para que asuma su rol. Tenga en cuenta que los requisitos para estar por fuera de las comillas en la cadena JSON pueden variar según su shell.

```
aws iam create-role --role-name lambda-ex --assume-role-policy-document '{"Version":
"2012-10-17","Statement": [{ "Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'
```

También puede definir la política de confianza para el rol con un archivo JSON aparte. En el siguiente ejemplo, `trust-policy.json` es un archivo que se encuentra en el directorio actual.

#### Example trust-policy.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
aws iam create-role --role-name lambda-ex --assume-role-policy-document file://trust-policy.json
```

Debería ver los siguientes datos de salida:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "lambda-ex",
    "RoleId": "AROAQFOXMPL6TZ6ITKWND",
    "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
    "CreateDate": "2020-01-17T23:19:12Z",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

```
}  
}
```

Para agregar permisos al rol, use el comando `attach-policy-to-role`. Los comandos siguientes agregan la política administrada `AWSLambdaBasicExecutionRole` al rol de ejecución `lambda-ex`.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/  
service-role/AWSLambdaBasicExecutionRole
```

Después de crear su rol de ejecución, adjúntelo a su función. Cuando [crea una función en la consola de Lambda](#), puede adjuntar a la función cualquier rol de ejecución que haya creado previamente. Si desea adjuntar un nuevo rol de ejecución a una función existente, siga los pasos que se indican en .

## Otorgue privilegios de acceso mínimos a su rol de ejecución de Lambda

Cuando crea por primera vez un rol de IAM para su función de Lambda durante la fase de desarrollo, a veces puede conceder permisos más allá de lo necesario. Antes de publicar la función en el entorno de producción, la práctica recomendada consiste en ajustar la política para incluir solo los permisos necesarios. Para obtener más información, consulte [Aplicar permisos de privilegio mínimo](#) en la Guía del usuario de IAM.

Utilice IAM Access Analyzer para ayudar a identificar los permisos necesarios para la política de rol de ejecución de IAM. IAM Access Analyzer revisa sus registros de AWS CloudTrail en el intervalo de fechas especificado y genera una plantilla de política con solo los permisos que la función utilizó durante ese tiempo. Puede utilizar la plantilla para crear una política administrada con permisos detallados y, a continuación, adjuntarla al rol de IAM. De esta forma, solo concede los permisos que el rol necesita para interactuar con los recursos de AWS para su caso de uso específico.

Para obtener más información, consulte [Generar políticas basadas en la actividad de acceso](#) en la Guía del usuario de IAM.

## Visualización y actualización de permisos en el rol de ejecución

En este tema se explica cómo ver y actualizar el [rol de ejecución de la función](#).

### Temas

- [Ver el rol de ejecución de una función](#)
- [Actualizar el rol de ejecución de una función](#)

## Ver el rol de ejecución de una función

Para ver el rol de ejecución de una función, utilice la consola de Lambda.

Cómo ver el rol de ejecución de una función (consola)

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija Configuration (Configuración) y, a continuación, seleccione Permissions (Permisos).
4. En Rol de ejecución, puede ver el rol que se utiliza actualmente como rol de ejecución de la función. Para mayor comodidad, puede ver todos los recursos y acciones a los que puede acceder la función en la sección de resumen de recursos. También puede elegir un servicio de la lista del menú desplegable para consultar los permisos relacionados con ese servicio.

## Actualizar el rol de ejecución de una función

Puede añadir o eliminar permisos del rol de ejecución de una función en cualquier momento, o configurar la función para que utilice otro rol. Si su función necesita acceder a otros servicios o recursos, debe agregar los permisos necesarios al rol de ejecución.

Cuando agregue permisos a su función, actualice también su código o configuración. Esto obliga a las instancias en ejecución de su función, cuyas credenciales han expirado, a detenerse y ser sustituidas.

Para actualizar el rol de ejecución de una función, puede utilizar la consola de Lambda.

Para configurar el rol de ejecución de la función (consola)

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija el nombre de una función.
3. Elija Configuration (Configuración) y, a continuación, seleccione Permissions (Permisos).
4. En Rol de ejecución, elija Editar.
5. Si desea actualizar su función para usar un rol diferente como rol de ejecución, elija el nuevo rol en el menú desplegable bajo el rol existente.

**Note**

Si desea actualizar los permisos de un rol de ejecución existente, solo puede hacerlo en la consola de AWS Identity and Access Management (IAM).

Si desea crear un nuevo rol para usarlo como rol de ejecución, elija Crear un nuevo rol a partir de plantillas de AWS políticas en el rol de ejecución. A continuación, introduzca un nombre para el rol nuevo en Nombre del rol y especifique las políticas que desee adjuntar al rol nuevo en Plantillas de políticas.

6. Seleccione Guardar.

## Trabajo con políticas administradas de AWS en el rol de ejecución

Las siguientes políticas administradas de AWS proporcionan los permisos necesarios para utilizar las características de Lambda.

Cambio	Descripción	Fecha
<a href="#">AWSLambdaMSKExecutionRole</a> : Lambda ha agregado el permiso <a href="#">kafka:DescribeClusterV2</a> a esta política.	AWSLambdaMSKExecutionRole concede permisos para leer y acceder a registros de un clúster de Amazon Managed Streaming for Apache Kafka (Amazon MSK), administrar interfaces de red elásticas (ENI) y escribir en CloudWatch Logs.	17 de junio de 2022
<a href="#">AWSLambdaBasicExecutionRole</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	AWSLambdaBasicExecutionRole concede permisos para cargar registros en CloudWatch.	14 de febrero de 2022
<a href="#">AWSLambdaDynamoDBExecutionRole</a> : Lambda	AWSLambdaDynamoDBExecutionRole concede	14 de febrero de 2022



Cambio	Descripción	Fecha
comenzó a realizar un seguimiento de los cambios de esta política.	permisos para leer registros de una transmisión de Amazon DynamoDB y escribir en CloudWatch Logs.	
<a href="#">AWSLambdaKinesisExecutionRole</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	<code>AWSLambdaKinesisExecutionRole</code> concede permisos para leer registros de una transmisión de datos de Amazon Kinesis y escribir en CloudWatch Logs.	14 de febrero de 2022
<a href="#">AWSLambdaMSKExecutionRole</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	<code>AWSLambdaMSKExecutionRole</code> concede permisos para leer y acceder a registros de un clúster de Amazon Managed Streaming for Apache Kafka (Amazon MSK), administrar interfaces de red elásticas (ENI) y escribir en CloudWatch Logs.	14 de febrero de 2022
<a href="#">AWSLambdaSQSQueueExecutionRole</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	<code>AWSLambdaSQSQueueExecutionRole</code> concede permisos para leer un mensaje de una cola de Amazon Simple Queue Service (Amazon SQS) y escribir en CloudWatch Logs.	14 de febrero de 2022
<a href="#">AWSLambdaVPCAccessExecutionRole</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	<code>AWSLambdaVPCAccessExecutionRole</code> concede permisos para administrar ENI dentro de una Amazon VPC y escribir en CloudWatch Logs.	14 de febrero de 2022

Cambio	Descripción	Fecha
<a href="#">AWSXRayDaemonWrite Access</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	AWSXRayDaemonWrite Access concede permisos para cargar datos de seguimiento en X-Ray.	14 de febrero de 2022
<a href="#">CloudWatchLambdaInsightsExecutionRolePolicy</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	CloudWatchLambdaInsightsExecutionRolePolicy concede permiso para escribir métricas de tiempo de ejecución en CloudWatch Lambda Insights.	14 de febrero de 2022
<a href="#">AmazonS3ObjectLambdaExecutionRolePolicy</a> : Lambda comenzó a realizar un seguimiento de los cambios de esta política.	AmazonS3ObjectLambdaExecutionRolePolicy concede permisos para interactuar con el objeto Lambda de Amazon Simple Storage Service (Amazon S3) y escribir en CloudWatch Logs.	14 de febrero de 2022

Para algunas características, la consola de Lambda intenta agregar permisos que faltan a su rol de ejecución en una política administrada por el cliente. Estas políticas pueden llegar a ser numerosas. Para evitar la creación de políticas adicionales, agregue las políticas administradas por AWS a su rol de ejecución antes de habilitar las características.

Cuando se utiliza un [mapeo de fuente de eventos](#) para invocar la función, Lambda utiliza el rol de ejecución para leer los datos de los eventos. Por ejemplo, un mapeo de fuente de eventos para Kinesis lee los eventos de un flujo de datos y se los envía a la función por lotes.

Cuando un servicio asume un rol en su cuenta, puede incluir las claves de contexto de condición global `aws:SourceAccount` y `aws:SourceArn` en la política de confianza de rol para limitar el acceso al rol a solo las solicitudes generadas por los recursos esperados. Para obtener más información, consulte [Prevención del suplente confuso entre servicios para AWS Security Token Service](#).

Además de las políticas administradas por AWS, la consola de Lambda proporciona plantillas para la creación de una política personalizada con permisos para casos de uso adicionales. Cuando crea una función en la consola de Lambda, puede elegir crear un nuevo rol de ejecución con permisos a partir de una o más plantillas. Estas plantillas también se aplican automáticamente cuando se crea una función a partir de un esquema o cuando se configuran opciones que requieren acceso a otros servicios. Existen plantillas de ejemplo en el [repositorio de GitHub](#) de esta guía.

## Uso del ARN de la función de origen para controlar el comportamiento de acceso a la función

Es común que el código de la función de Lambda realice solicitudes de API a otros servicios de AWS. Para realizar estas solicitudes, Lambda genera un conjunto efímero de credenciales al asumir el rol de ejecución de la función. Estas credenciales están disponibles como variables de entorno durante la invocación de la función. Cuando trabaja con el SDK de AWS, no es necesario proporcionar las credenciales de SDK directamente en el código. De forma predeterminada, la cadena de proveedores de credenciales comprueba secuencialmente cada lugar en el que puede configurar las credenciales y elige el primero disponible, normalmente las variables de entorno (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, y `AWS_SESSION_TOKEN`).

Lambda inyecta el ARN de la función de origen en el contexto de credenciales solo si la solicitud es una solicitud de la API de AWS que proviene de su entorno de ejecución. Lambda también inyecta el ARN de la función de origen para las siguientes solicitudes a la API de AWS que Lambda realiza fuera del entorno de ejecución en su nombre:

Servicio	Acción	Motivo
Registros de CloudWatch	<code>CreateLogGroup</code> , <code>CreateLogStream</code> , <code>PutLogEvents</code>	Para almacenar los registros en un grupo de registro de los registros de CloudWatch
X-Ray	<code>PutTraceSegments</code>	Para enviar datos de seguimiento a X-Ray
Amazon EFS	<code>ClientMount</code>	Para conectar la función a un sistema de archivos Amazon Elastic File System (Amazon EFS)

Las llamadas a la API de AWS que realiza fuera del entorno de ejecución en su nombre con el mismo rol de ejecución no contienen el ARN de la función de origen. Entre los ejemplos de llamadas a la API fuera del entorno de ejecución se incluyen los siguientes:

- Llamadas a AWS Key Management Service (AWS KMS) para cifrar y descifrar de forma automática las variables de entorno.
- Llamadas a Amazon Elastic Compute Cloud (Amazon EC2) para crear interfaces de red elásticas (ENI) para una función con VPC habilitada.
- Llama a servicios de AWS, como Amazon Simple Queue Service (Amazon SQS), para leer desde un origen de eventos que está configurada como una [asignación de orígenes de eventos](#).

Con el ARN de la función de origen en el contexto de credenciales, puede verificar si una llamada al recurso proviene del código de una función de Lambda específica. Para comprobarlo, utilice la clave de condición `lambda:SourceFunctionArn` en una política con base en identidad de IAM o [política de control de servicio \(SCP\)](#).

#### Note

No puede utilizar la clave de condición `lambda:SourceFunctionArn` en las políticas basadas en recursos.

Con esta clave de condición en las políticas basadas en la identidad o SCP, puede implementar controles de seguridad para las acciones de la API que el código de la función realiza en otros servicios de AWS. Tiene algunas aplicaciones de seguridad clave, como ayudarlo a identificar el origen de una fuga de credenciales.

#### Note

La clave de condición `lambda:SourceFunctionArn` es diferente de las claves de condición `lambda:FunctionArn` y `aws:SourceArn`. La clave de condición `lambda:FunctionArn` solo se aplica a [asignaciones de orígenes de eventos](#) y ayuda a definir qué funciones puede invocar el origen de eventos. La clave de condición `aws:SourceArn` se aplica solo a las políticas en las que la función de Lambda es el recurso de destino y ayuda a definir qué otros servicios y los recursos de AWS pueden invocar esa función. La clave de condición `lambda:SourceFunctionArn` se puede aplicar a cualquier

política basada en identidad o SCP para definir las funciones Lambda específicas que tienen permisos para hacer llamadas de la API de AWS a otros recursos.

Para utilizar `lambda:SourceFunctionArn` en la política, inclúyala como condición en cualquiera de los [operadores de condición ARN](#). El valor de la clave debe ser un ARN válido.

Por ejemplo, suponga que el código de función de Lambda hace una llamada `s3:PutObject` que está dirigida a un bucket de Amazon S3 específico. Es posible que desee permitir que solo una función de Lambda específica tenga acceso a `s3:PutObject` de ese bucket. En este caso, el rol de ejecución de la función debe tener una política asociada similar a la siguiente:

Example política que otorga acceso a una función de Lambda específica a un recurso de Amazon S3

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleSourceFunctionArn",
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Condition": {
        "ArnEquals": {
          "lambda:SourceFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:source_lambda"
        }
      }
    }
  ]
}
```

Esta política solo permite el acceso a `s3:PutObject` si el origen es la función de Lambda con el ARN `arn:aws:lambda:us-east-1:123456789012:function:source_lambda`. Esta política no permite el acceso a `s3:PutObject` de ninguna otra identidad de llamada. Esto se aplica incluso aunque una función o entidad diferente haga una llamada `s3:PutObject` con el mismo rol de ejecución.

**Note**

La clave de condición `lambda:SourceFunctionARN` no admite versiones de funciones de Lambda ni alias de funciones. Si usa el ARN para una versión o alias de función en particular, la función no tendrá permiso para realizar la acción que especifique. Asegúrese de utilizar el ARN incompleto para la función sin un sufijo de versión o alias.

También se puede usar `lambda:SourceFunctionArn` en SCP. Por ejemplo, suponga que desea restringir el acceso al bucket a un solo código de la función de Lambda o a llamadas de una instancia específica de Amazon Virtual Private Cloud (VPC). La siguiente SCP ilustra este caso.

Example política que deniega el acceso a Amazon S3 en condiciones específicas

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:*"
      ],
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Effect": "Deny",
      "Condition": {
        "StringNotEqualsIfExists": {
          "aws:SourceVpc": [
            "vpc-12345678"
          ]
        }
      }
    },
    {
      "Action": [
        "s3:*"
      ],
      "Resource": "arn:aws:s3:::lambda_bucket/*",
      "Effect": "Deny",
      "Condition": {
        "ArnNotEqualsIfExists": {
          "lambda:SourceFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:source_lambda"
        }
      }
    }
  ]
}
```

```
}
  }
]
}
```

Esta política deniega todas las acciones de S3 a menos que provengan de una función de Lambda específica con el ARN `arn:aws:lambda:*:123456789012:function:source_lambda` o de la VPC especificada. El operador `StringNotEqualsIfExists` indica a IAM que procese esta condición solo si la clave `aws:SourceVpc` está presente en la solicitud. Del mismo modo, IAM considera el operador `ArnNotEqualsIfExists` solo si `lambda:SourceFunctionArn` está presente.

# Otorgar acceso a otras entidades de AWS a sus funciones de Lambda

Para conceder permisos para que otras Cuentas de AWS, organizaciones y servicios accedan a sus recursos de Lambda, tiene varias opciones:

- Puede utilizar [políticas basadas en identidades](#) para conceder acceso a otros usuarios a sus recursos de Lambda. Las políticas basadas en identidad se pueden aplicar a los usuarios directamente, o a los grupos y roles que están asociados a un usuario.
- Puede usar las [políticas basadas en recursos](#) para darles permiso a otras cuentas y servicios de AWS para acceder a los recursos de Lambda. Cuando un usuario intenta acceder a un recurso de Lambda, este servicio tiene en cuenta tanto las políticas basadas en identidades como la política basada en recursos del recurso. Cuando un servicio de AWS, como Amazon Simple Storage Service (Amazon S3), llama a la función de Lambda, el servicio considera solo la política basada en recursos.
- Puede utilizar un modelo de [control de acceso basado en atributos \(ABAC\)](#) para controlar el acceso a las funciones de Lambda. Con ABAC, puede adjuntar etiquetas a una función de Lambda, pasarlas en determinadas solicitudes de la API o adjuntarlas a la entidad principal de IAM que realiza la solicitud. Especifique las mismas etiquetas en el elemento de condición de una política de IAM para controlar el acceso a la función.

Para ayudarlo a ajustar sus permisos para el acceso con privilegios mínimos, Lambda proporciona algunas condiciones adicionales que puede incluir en sus políticas. Para obtener más información, consulte [the section called “Recursos y condiciones”](#).

## Políticas de IAM basadas en identidad para Lambda

Puede utilizar las políticas basadas en identidad de AWS Identity and Access Management (IAM) para conceder a los usuarios de su cuenta acceso a Lambda. Las políticas basadas en identidad se pueden aplicar a usuarios, grupos de usuarios o roles. También puede conceder a los usuarios de otra cuenta permiso para asumir un rol de su cuenta y tener acceso a sus recursos de Lambda.

Lambda proporciona a políticas administradas AWS que otorgan acceso a las acciones de la API de Lambda y, en algunos casos, acceso a otros servicios de AWS utilizados para desarrollar y administrar los recursos de Lambda. Lambda actualiza las políticas administradas según sea necesario para garantizar que los usuarios tengan acceso a las características nuevas que se publiquen.



- [AWSLambda\\_FullAccess](#): concede acceso completo a las acciones de Lambda y a otros servicios AWS que se utilizan para desarrollar y mantener los recursos de Lambda.
- [AWSLambda\\_ReadOnlyAccess](#): concede acceso de solo lectura a los recursos de Lambda.
- [AWSLambdaRole](#): concede permisos para invocar funciones de Lambda.

Las políticas administradas AWS conceden permiso a las acciones de la API sin restringir las funciones de Lambda ni las capas que un usuario puede modificar. Para conseguir un control más preciso, puede crear sus propias políticas que limiten el ámbito de los permisos de un usuario.

## Temas

- [Concesión de acceso a una función de Lambda a los usuarios](#)
- [Conceder a los usuarios acceso a una capa de Lambda](#)

## Concesión de acceso a una función de Lambda a los usuarios

Utilice [políticas basadas en identidad](#) para permitir a los usuarios, los grupos de usuarios o los roles realizar operaciones en las funciones de Lambda.

### Note

Para una función definida como imagen de contenedor, el permiso de usuario para acceder a la imagen debe configurarse en Amazon Elastic Container Registry (Amazon ECR). Para ver un ejemplo, consulte [Políticas de repositorios de Amazon ECR](#).

A continuación, se muestra un ejemplo de una política de permisos con un ámbito limitado. Permite a un usuario crear y administrar funciones de Lambda cuyo nombre tiene un prefijo determinado (`intern-`) y que están configuradas con un rol de ejecución determinado.

### Example Política para el desarrollo de funciones

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
```

```

        "lambda:GetAccountSettings",
        "lambda:GetEventSourceMapping",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration",
        "lambda:GetFunctionCodeSigningConfig",
        "lambda:GetFunctionConcurrency",
        "lambda>ListEventSourceMappings",
        "lambda>ListFunctions",
        "lambda>ListTags",
        "iam>ListRoles"
    ],
    "Resource": "*"
},
{
    "Sid": "DevelopFunctions",
    "Effect": "Allow",
    "NotAction": [
        "lambda:AddPermission",
        "lambda:PutFunctionConcurrency"
    ],
    "Resource": "arn:aws:lambda:*:*:function:intern-*"
},
{
    "Sid": "DevelopEventSourceMappings",
    "Effect": "Allow",
    "Action": [
        "lambda>DeleteEventSourceMapping",
        "lambda:UpdateEventSourceMapping",
        "lambda>CreateEventSourceMapping"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
        }
    }
},
{
    "Sid": "PassExecutionRole",
    "Effect": "Allow",
    "Action": [
        "iam>ListRolePolicies",
        "iam>ListAttachedRolePolicies",
        "iam:GetRole",

```

```

        "iam:GetRolePolicy",
        "iam:PassRole",
        "iam:SimulatePrincipalPolicy"
    ],
    "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
},
{
    "Sid": "ViewLogs",
    "Effect": "Allow",
    "Action": [
        "logs:*"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
}
]
}

```

Los permisos de la política están organizados en instrucciones que se basan en las [condiciones y los recursos](#) a los que se aplican.

- **ReadOnlyPermissions:** la consola de Lambda utiliza estos permisos cuando se buscan y se visualizan las funciones. No admiten patrones de recursos ni condiciones.

```

    "Action": [
        "lambda:GetAccountSettings",
        "lambda:GetEventSourceMapping",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration",
        "lambda:GetFunctionCodeSigningConfig",
        "lambda:GetFunctionConcurrency",
        "lambda>ListEventSourceMappings",
        "lambda>ListFunctions",
        "lambda>ListTags",
        "iam>ListRoles"
    ],
    "Resource": "*"

```

- **DevelopFunctions:** usa cualquier acción de Lambda que opera sobre las funciones cuyo nombre tiene el prefijo `intern-`, excepto `AddPermission` y `PutFunctionConcurrency`. `AddPermission` modifica la [política basada en recursos](#) de la función y puede afectar a la

seguridad. `PutFunctionConcurrency` reserva capacidad de escalado para una función y puede quitar capacidad a las otras funciones.

```
"NotAction": [
  "lambda:AddPermission",
  "lambda:PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*:*:function:intern-*
```

- `DevelopEventSourceMappings`: permite administrar las asignaciones de orígenes de eventos en las funciones cuyo nombre tenga el prefijo `intern-`. Estas acciones operan sobre los mapeos de orígenes de eventos, pero es posible restringirlas para las distintas funciones utilizando una condición.

```
"Action": [
  "lambda>DeleteEventSourceMapping",
  "lambda:UpdateEventSourceMapping",
  "lambda>CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
  "StringLike": {
    "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
  }
}
```

- `PassExecutionRole`: permite ver y pasar únicamente un rol denominado `intern-lambda-execution-role`, que debe crear y administrar un usuario con permisos de IAM. `PassRole` se utiliza cuando se asigna un rol de ejecución a una función.

```
"Action": [
  "iam:ListRolePolicies",
  "iam:ListAttachedRolePolicies",
  "iam:GetRole",
  "iam:GetRolePolicy",
  "iam:PassRole",
  "iam:SimulatePrincipalPolicy"
],
```

```
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- **ViewLogs:** permite utilizar CloudWatch Logs para ver los registros de las funciones cuyo nombre tiene el prefijo intern-.

```
"Action": [
  "logs:*"
],
"Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Esta política permite a un usuario comenzar a utilizar Lambda sin poner en peligro los recursos de los demás usuarios. No permite a un usuario configurar una función que llame a o que se active por otros servicios de AWS, lo cual requiere permisos de IAM más amplios. Tampoco incluye permiso para los servicios que no admiten políticas de ámbito limitado, como CloudWatch y X-Ray. Utilice las políticas de solo lectura para estos servicios con objeto de dar a los usuarios acceso a las métricas y los datos de rastreo.

Al configurar triggers para una función, se necesita acceso para utilizar el servicio AWS que invoca la función. Por ejemplo, si desea configurar un desencadenador de Amazon S3, necesita permiso para poder utilizar las acciones de Amazon S3 que permiten administrar notificaciones de buckets. Muchos de estos permisos se incluyen en la política administrada [AWSLambda\\_FullAccess](#).

## Conceder a los usuarios acceso a una capa de Lambda

Utilice [políticas basadas en identidad](#) para permitir a los usuarios realizar operaciones en las funciones de Lambda. La siguiente política concede a un usuario permiso para crear capas y usarlas con las funciones. Los patrones de recursos permiten al usuario trabajar en cualquier Región de AWS y con cualquier versión de capa, siempre y cuando el nombre de la capa comience con test-.

### Example Política de desarrollo de capas

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublishLayers",
      "Effect": "Allow",
      "Action": [
        "lambda:PublishLayerVersion"
      ]
    }
  ]
}
```

```

    ],
    "Resource": "arn:aws:lambda:*:*:layer:test-*"
  },
  {
    "Sid": "ManageLayerVersions",
    "Effect": "Allow",
    "Action": [
      "lambda:GetLayerVersion",
      "lambda>DeleteLayerVersion"
    ],
    "Resource": "arn:aws:lambda:*:*:layer:test-*:*"
  }
]
}

```

También puede hacer obligatorio el uso de capas durante la creación y configuración de funciones con la condición `lambda:Layer`. Por ejemplo, puede impedir que los usuarios utilicen capas publicados por otras cuentas. La siguiente política añade una condición a las acciones `CreateFunction` y `UpdateFunctionConfiguration` para exigir que las capas especificadas procedan de la cuenta `123456789012`.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ConfigureFunctions",
      "Effect": "Allow",
      "Action": [
        "lambda>CreateFunction",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "lambda:Layer": [
            "arn:aws:lambda:*:123456789012:layer:*:*"
          ]
        }
      }
    }
  ]
}

```

Para asegurarse de que se aplica la condición, compruebe que ninguna otra instrucción conceda al usuario permiso para estas acciones.

## Trabajar con políticas de IAM basadas en recursos en Lambda

Lambda admite políticas de permisos basadas en recursos para las funciones y capas de Lambda. Puede usar políticas basadas en recursos para conceder acceso a otras [cuentas de AWS](#), [organizaciones](#) o [servicios](#). Las políticas basadas en recursos se aplican a una sola función, versión, alias o versión de capa.

### Console

Visualización de la política basada en recursos de una función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuración y, a continuación, seleccione Permisos.
4. Desplácese hacia abajo hasta Directiva basada en recursos y, a continuación, elija Ver documento de directiva. La política basada en recursos muestra los permisos que se aplican cuando otra cuenta o servicio de AWS intenta acceder a la función. En el ejemplo siguiente se muestra una instrucción que permite a Amazon S3 invocar una función denominada `my-function` para un bucket denominado `amzn-s3-demo-bucket` en la cuenta `123456789012`.

### Example Política basada en recursos

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "lambda-allow-s3-my-function",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-
function",
      "Condition": {
```

```

        "StringEquals": {
            "AWS:SourceAccount": "123456789012"
        },
        "ArnLike": {
            "AWS:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    }
}
]
}

```

## AWS CLI

Para ver la política basada en recursos de una función, utilice el comando `get-policy`.

```

aws lambda get-policy \
  --function-name my-function \
  --output text

```

Debería ver los siguientes datos de salida:

```

{"Version":"2012-10-17","Id":"default","Statement":
[{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"s3.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function","Condition":{"ArnLike":
{"AWS:SourceArn":"arn:aws:sns:us-east-2:123456789012:lambda*"}}}]}      7c681fc9-
b791-4e91-acdf-eb847fdaa0f0

```

Para las versiones y los alias, añada el número de versión o el alias al nombre de la función.

```

aws lambda get-policy --function-name my-function:PROD

```

Para eliminar permisos de una función, utilice `remove-permission`.

```

aws lambda remove-permission \
  --function-name example \
  --statement-id sns

```

Utilice el comando `get-layer-version-policy` para ver los permisos de una capa.



```
aws lambda get-layer-version-policy \  
  --layer-name my-layer \  
  --version-number 3 \  
  --output text
```

Debería ver los siguientes datos de salida:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236    {"Sid":"engineering-  
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:  
west-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":  
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}
```

Se utiliza `remove-layer-version-permission` para quitar instrucciones de la política.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3  
  --statement-id engineering-org
```

## Acciones de la API admitidas

Las siguientes acciones de la API de Lambda admiten las políticas basadas en recursos:

- [CreateAlias](#)
- [DeleteAlias](#)
- [DeleteFunction](#)
- [DeleteFunctionConcurrency](#)
- [DeleteFunctionEventInvokeConfig](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [GetAlias](#)
- [GetFunction](#)
- [GetFunctionConcurrency](#)
- [GetFunctionConfiguration](#)
- [GetFunctionEventInvokeConfig](#)
- [GetPolicy](#)
- [GetProvisionedConcurrencyConfig](#)

- [Invoke](#)
- [ListAliases](#)
- [ListFunctionEventInvokeConfigs](#)
- [ListProvisionedConcurrencyConfigs](#)
- [ListTags](#)
- [ListVersionsByFunction](#)
- [PublishVersion](#)
- [PutFunctionConcurrency](#)
- [PutFunctionEventInvokeConfig](#)
- [PutProvisionedConcurrencyConfig](#)
- [TagResource](#)
- [UntagResource](#)
- [UpdateAlias](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionEventInvokeConfig](#)

## Concesión de acceso a las funciones de Lambda a los servicios de AWS

Cuando se [utiliza un servicio de AWS para invocar una función](#), se debe conceder permiso en una instrucción de una política basada en recursos. Puede aplicar la instrucción a toda la función, o bien, puede limitar la instrucción a una sola versión o alias.

### Note

Cuando se añade un desencadenador a la función con la consola de Lambda, esta actualiza la política basada en recursos de la función para permitir al servicio que la invoque. Para conceder permisos a otras cuentas o servicios que no están disponibles en la consola de Lambda, puede utilizar la AWS CLI.

Agregue una instrucción con el comando [add-permission](#). La instrucción más sencilla de una política basada en recursos permite un servicio invocar una función. El siguiente comando concede a Amazon Simple Notification Service permiso para invocar una función denominada `my-function`.

```
aws lambda add-permission \  
  --function-name my-function \  
  --action lambda:InvokeFunction \  
  --statement-id sns \  
  --principal sns.amazonaws.com \  
  --output text
```

Debería ver los siguientes datos de salida:

```
{"Sid":"sns","Effect":"Allow","Principal":  
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-  
east-2:123456789012:function:my-function"}
```

De este modo, Amazon SNS podrá llamar la acción [Invoke](#) de la API para la función, pero no se restringirá el tema de Amazon SNS que activa la invocación. Para asegurarse de que la función solo la invocada un recurso determinado, especifique el nombre de recurso de Amazon (ARN) del recurso con la opción `source-arn`. El siguiente comando solo permite a Amazon SNS invocar la función para las suscripciones a un tema denominado `my-topic`.

```
aws lambda add-permission \  
  --function-name my-function \  
  --action lambda:InvokeFunction \  
  --statement-id sns-my-topic \  
  --principal sns.amazonaws.com \  
  --source-arn arn:aws:sns:us-east-2:123456789012:my-topic
```

Algunos servicios pueden invocar funciones de otras cuentas. Esto no es un problema cuando se especifica un ARN de origen que incluye el ID de la cuenta. Sin embargo, para Amazon S3, la fuente es un bucket cuyo ARN no incluye el ID de la cuenta. Es posible que usted elimine el bucket y que otra cuenta cree un bucket con el mismo nombre. Utilice la opción `source-account` con el ID de la cuenta para garantizar que solo los recursos de la cuenta pueden invocar la función.

```
aws lambda add-permission \  
  --function-name my-function \  
  --action lambda:InvokeFunction \  
  --statement-id s3-account \  
  --principal s3.amazonaws.com \  
  --source-arn arn:aws:s3::amzn-s3-demo-bucket \  
  --source-account 123456789012
```

## Concesión de acceso a las funciones a una organización

Para conceder permisos a una organización definida en [AWS Organizations](#), especifique el ID de la organización como el `principal-org-id`. El siguiente comando [add-permission](#) concede acceso de invocación a todos los usuarios de la organización `o-a1b2c3d4e5f`.

```
aws lambda add-permission \  
  --function-name example \  
  --statement-id PrincipalOrgIDExample \  
  --action lambda:InvokeFunction \  
  --principal * \  
  --principal-org-id o-a1b2c3d4e5f
```

### Note

En este comando, `Principal` es `*`. Esto significa que todos los usuarios de la organización `o-a1b2c3d4e5f` obtienen permisos de invocación de funciones. Si especifica una Cuenta de AWS o un rol como `Principal`, entonces solo esa entidad principal obtiene permisos de invocación de funciones, pero solo si también forman parte de la organización `o-a1b2c3d4e5f`.

Este comando crea una política basada en recursos que tiene el siguiente aspecto:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "PrincipalOrgIDExample",  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": "lambda:InvokeFunction",  
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:example",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalOrgID": "o-a1b2c3d4e5f"  
        }  
      }  
    }  
  ]  
}
```

```
}
```

Para obtener más información, consulte [aws:PrincipalOrgID](#) en la Guía del usuario de IAM.

## Concesión de acceso a las funciones de Lambda a otras cuentas

Para compartir una función con otra Cuenta de AWS, agregue una declaración de permisos entre cuentas a la [política basada en los recursos](#) de la función. Ejecute el comando [add-permission](#) y especifique el ID de la cuenta como `principal`. En el siguiente ejemplo, se concede permiso a la cuenta 111122223333 para invocar `my-function` con el alias `prod`.

```
aws lambda add-permission \  
  --function-name my-function:prod \  
  --statement-id xaccount \  
  --action lambda:InvokeFunction \  
  --principal 111122223333 \  
  --output text
```

Debería ver los siguientes datos de salida:

```
{"Sid":"xaccount","Effect":"Allow","Principal":  
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-1:123456789012:function:my-function"}
```

La política basada en recursos concede permiso a la otra cuenta para obtener acceso a la función, pero no permite a los usuarios de esa cuenta superar sus permisos. Los usuarios de la otra cuenta deben tener los [permisos de usuario](#) correspondientes para utilizar la API de Lambda.

Para limitar el acceso a un usuario o función de otra cuenta, especifique el ARN completo de la identidad como el `principal`. Por ejemplo, `arn:aws:iam::123456789012:user/developer`.

El [alias](#) limita la versión que puede invocar la otra cuenta. Requiere que la otra cuenta incluya el alias en el ARN de la función.

```
aws lambda invoke \  
  --function-name arn:aws:lambda:us-east-2:123456789012:function:my-function:prod out
```

Debería ver los siguientes datos de salida:

```
{  
  "StatusCode": 200,
```

```
"ExecutedVersion": "1"
}
```

El propietario de la función puede actualizar el alias para que apunte a una nueva versión sin que el autor de la llamada tenga que cambiar la forma en que invoca la función. De este modo, no es necesario que la otra cuenta cambie su código para utilizar la nueva versión y se garantiza que solo tiene permiso para invocar la versión de la función asociada al alias.

Puede conceder acceso entre cuentas para la mayoría de acciones de la API que [operen con una función existente](#). Por ejemplo, puede conceder acceso a `lambda:ListAliases` para permitir que una cuenta obtenga una lista de alias o a `lambda:GetFunction` para permitirles que descarguen el código de la función. Añada cada permiso por separado o utilice `lambda:*` para conceder acceso a todas las acciones para la función especificada.

Si desea conceder a otras cuentas permiso sobre varias funciones o sobre acciones que no pueden utilizarse con una función, le recomendamos que utilice los [roles de IAM](#).

## Concesión de acceso a las capas de Lambda a otras cuentas

Para compartir una capa con otra Cuenta de AWS, agregue una instrucción de permisos para varias cuentas a la [política basada en los recursos](#) de la capa. Ejecute el comando [add-layer-version-permission](#) y especifique el ID de cuenta como `principal`. En cada instrucción, puede conceder permiso a una única cuenta, a todas las cuentas o a una organización en [AWS Organizations](#).

El siguiente ejemplo concede a la cuenta 111122223333 acceso a la versión 2 de la capa `bash-runtime`.

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Debería ver una salida similar a esta:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Los permisos solo se aplican a una única versión de una capa. Repita el proceso cada vez que cree una nueva versión de la capa.

Para conceder permiso a todas las cuentas en una organización de [AWS Organizations](#), utilice la opción `organization-id`. En el siguiente ejemplo, se concede a todas las cuentas de una organización `o-t194hfs8cz` permiso para utilizar la versión 3 de `my-layer`.

```
aws lambda add-layer-version-permission \  
  --layer-name my-layer \  
  --version-number 3 \  
  --statement-id engineering-org \  
  --principal '*' \  
  --action lambda:GetLayerVersion \  
  --organization-id o-t194hfs8cz \  
  --output text
```

Debería ver los siguientes datos de salida:

```
{"Sid":"engineering-  
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam  
east-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":  
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}
```

Para conceder permisos a diferentes cuentas u organizaciones, tendrá que agregar varias instrucciones.

## Control de acceso basado en atributos para Lambda

Con el [control de acceso basado en atributos \(ABAC\)](#), puede utilizar etiquetas para controlar el acceso a las funciones de Lambda. Puede adjuntar etiquetas a una función de Lambda, pasarlas en determinadas solicitudes de API o adjuntarlas a la entidad principal de AWS Identity and Access Management (IAM) que realiza la solicitud. Para obtener más información sobre cómo AWS concede el acceso basado en atributos, consulte [Controlar el acceso a los recursos de AWS mediante etiquetas](#) en la Guía del usuario de IAM.

Puede usar ABAC para [conceder el privilegio mínimo](#) sin especificar un nombre de recurso de Amazon (ARN) o un patrón de ARN en la política de IAM. En su lugar, puede especificar una etiqueta en el [elemento de condición](#) de una política de IAM para controlar el acceso. El escalado es más fácil con ABAC porque no tiene que actualizar las políticas de IAM cuando crea nuevas funciones. En cambio, agregue etiquetas a las nuevas funciones para controlar el acceso.

En Lambda, las etiquetas funcionan a nivel de función. Las etiquetas no se admiten en capas, configuraciones de firma de código, ni asignaciones de orígenes de eventos. Al etiquetar una función, esas etiquetas se aplican a todas las versiones y alias asociados a la función. Para obtener más información sobre cómo etiquetar funciones, consulte [Uso de etiquetas en funciones de Lambda](#).

Puede utilizar las siguientes claves de condición para controlar las acciones de las funciones:

- [aws:ResourceTag/tag-key](#): controle el acceso en función de las etiquetas que se adjuntan a las funciones de Lambda.
- [aws:RequestTag/tag-key](#): solicite que las etiquetas estén presentes en una solicitud, por ejemplo, al crear una nueva función.
- [aws:PrincipalTag/tag-key](#): controle lo que la entidad principal de IAM (la persona que hace la solicitud) está autorizada a hacer en función de las etiquetas que se adjuntan a su [usuario](#) o [rol](#) de IAM.
- [aws:TagKeys](#): controle si se pueden utilizar claves de etiquetas específicas en una solicitud.

Para obtener una lista completa de las acciones de Lambda que admite ABAC, consulte [Acciones de función admitidas](#) y verifique la columna Condition (Condición) de la tabla.

Los siguientes pasos muestran una forma de configurar permisos mediante ABAC. En este escenario de ejemplo, creará cuatro políticas de permisos de IAM. A continuación, adjuntará estas políticas a un nuevo rol de IAM. Por último, creará un usuario de IAM y le concederá permiso para que asuma el nuevo rol.

## Temas

- [Requisitos previos](#)
- [Paso 1: solicitar etiquetas en las nuevas funciones](#)
- [Paso 2: permitir acciones basadas en etiquetas adjuntas a una función de Lambda y a una entidad principal de IAM](#)
- [Paso 3: conceder permisos de lista](#)
- [Paso 4: conceder permisos de IAM](#)
- [Paso 5: crear el rol de IAM](#)
- [Paso 6: crear el usuario de IAM](#)
- [Paso 7: probar los permisos](#)
- [Paso 5: Eliminar los recursos](#)



## Requisitos previos

Asegúrese de que tiene un [rol de ejecución de Lambda](#). Utilizará este rol al conceder permisos de IAM y al crear una función de Lambda.

### Paso 1: solicitar etiquetas en las nuevas funciones

Cuando se usa ABAC con Lambda, es una práctica recomendada solicitar que todas las funciones tengan etiquetas. Esto ayuda a garantizar que las políticas de permisos de ABAC funcionen según lo esperado.

[Cree una política de IAM](#) similar al siguiente ejemplo: Esta política utiliza las claves de condición [aws:RequestTag/tag-key](#), [aws:ResourceTag/tag-key](#) y [aws:TagKeys](#) para solicitar que las funciones nuevas y la entidad principal de IAM que crea las funciones tengan la etiqueta `project`. El modificador `ForAllValues` garantiza que `project` sea la única etiqueta permitida. Si no incluye el modificador `ForAllValues`, los usuarios pueden agregar otras etiquetas a la función siempre que también pasen `project`.

#### Example — Solicitar etiquetas en las nuevas funciones

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "lambda:CreateFunction",
      "lambda:TagResource"
    ],
    "Resource": "arn:aws:lambda:*:*:function:*",
    "Condition": {
      "StringEquals": {
        "aws:RequestTag/project": "${aws:PrincipalTag/project}",
        "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
      },
      "ForAllValues:StringEquals": {
        "aws:TagKeys": "project"
      }
    }
  }
}
```

## Paso 2: permitir acciones basadas en etiquetas adjuntas a una función de Lambda y a una entidad principal de IAM

Describa cómo crear una segunda política de IAM mediante la clave de condición [aws:ResourceTag/tag-key](#) para solicitar que la etiqueta de la entidad principal coincida con la etiqueta adjunta a la función. El siguiente ejemplo de política permite a las entidades principales con la etiqueta `project` invocar funciones con la etiqueta `project`. Si una función tiene otras etiquetas, se deniega la acción.

### Example — Solicitar etiquetas coincidentes en la función y la entidad principal de IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:GetFunction"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
        }
      }
    }
  ]
}
```

## Paso 3: conceder permisos de lista

Cree una política que permita a la entidad principal enumerar las funciones de Lambda y los roles de IAM. Esto permite a la entidad principal ver todas las funciones de Lambda y los roles de IAM en la consola y cuando llama a las acciones de la API.

### Example — Conceder permisos de lista de Lambda e IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Sid": "AllResourcesLambdaNoTags",
    "Effect": "Allow",
    "Action": [
        "lambda:GetAccountSettings",
        "lambda:ListFunctions",
        "iam:ListRoles"
    ],
    "Resource": "*"
  }
]
}

```

## Paso 4: conceder permisos de IAM

Cree una política que permita `iam:PassRole`. Este permiso es necesario al asignar un rol de ejecución a una función. En la siguiente política de ejemplo, reemplace el ARN de ejemplo por el ARN del rol de ejecución de Lambda.

### Note

No utilice la clave de condición de `ResourceTag` en una política con la acción `iam:PassRole`. No puede utilizar la etiqueta en un rol de IAM para controlar el acceso a quién puede pasar ese rol. Para obtener más información sobre los permisos necesarios a fin de pasar un rol a un servicio, consulte [Concesión de permisos a un usuario para pasar un rol a un servicio de AWS](#).

### Example — Conceder permiso para pasar el rol de ejecución

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
    }
  ]
}

```

```
}
```

## Paso 5: crear el rol de IAM

Es una práctica recomendada [utilizar roles para delegar permisos](#). [Cree un rol de IAM](#) denominado `abac-project-role`:

- En Step 1: Select trusted entity (Paso 1: Seleccionar una entidad de confianza): seleccione AWS account (Cuenta de ) y luego This account (Esta cuenta).
- En Step 2: Add permissions (Paso 2: agregar permisos): adjunte las cuatro políticas de IAM que creó en los pasos anteriores.
- En Step 3: Name, review, and create (Paso 3: nombrar, revisar y crear): elija Add tag (Agregar etiqueta). En Clave, escriba `project`. No ingrese nada en Value (Valor).

## Paso 6: crear el usuario de IAM

[Cree un usuario de IAM](#) denominado `abac-test-user`. En la sección Set permissions (Establecer permisos), elija Attach existing policies directly (Adjuntar directamente las políticas existentes) y, a continuación, Create policy (Crear política). Escriba la siguiente definición de política. Reemplace `111122223333` por su [ID de cuenta de AWS](#). Esta política permite al `abac-test-user` asumir el `abac-project-role`.

Example — Permitir que el usuario de IAM asuma el rol de ABAC

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
  }
}
```

## Paso 7: probar los permisos

1. Inicie sesión en la consola de AWS como `abac-test-user`. Para obtener más información, consulte [Iniciar sesión como usuario de IAM](#).
2. Cambie al rol `abac-project-role`. Para obtener más información, consulte [Cambiar a un rol \(consola\)](#).

### 3. [Cree una función de Lambda:](#)

- En Permissions (Permisos), elija Change default execution role (Cambiar rol de ejecución predeterminado) y, a continuación, en Execution role (Rol de ejecución), elija Use an existing role (Usar un rol existente). Elija el mismo rol de ejecución que utilizó en [Paso 4: conceder permisos de IAM](#).
- En Advanced settings (Configuración avanzada), elija Enable tags (Habilitar etiquetas) y, a continuación, Add new tag (Agregar nueva etiqueta). En Clave, escriba project. No ingrese nada en Value (Valor).

### 4. [Pruebe la función.](#)

5. Cree una segunda función de Lambda y agregue una etiqueta diferente, como environment. Esta operación puede fallar porque la política de ABAC que creó en [Paso 1: solicitar etiquetas en las nuevas funciones](#) solo permite a la entidad principal crear funciones con la etiqueta project.
6. Cree una tercera función sin etiquetas. Esta operación puede fallar porque la política de ABAC que creó en [Paso 1: solicitar etiquetas en las nuevas funciones](#) solo permite a la entidad principal crear funciones sin etiquetas.

Esta estrategia de autorización permite controlar el acceso sin crear nuevas políticas para cada usuario nuevo. Para conceder acceso a los nuevos usuarios, basta con darles permiso a fin de que asuman el rol que corresponde a su proyecto asignado.

## Paso 5: Eliminar los recursos

Para eliminar el rol de IAM

1. Abra la [página Roles](#) en la consola de IAM.
2. Seleccione el rol que creó en el [paso 5](#).
3. Elija Eliminar.
4. Para confirmar la eliminación, escriba el nombre del rol en el campo de entrada de texto.
5. Elija Eliminar.

Cómo eliminar el usuario de IAM

1. Abra la [página Users](#) en la consola de IAM;
2. Seleccione el usuario de IAM que ha creado en el [paso 6](#).

3. Elija Eliminar.
4. Para confirmar la eliminación, escriba el nombre del usuario en el campo de entrada de texto.
5. Elija Eliminar usuario.

### Cómo eliminar la función de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione la función que ha creado.
3. Elija Acciones, Eliminar.
4. Escriba **delete** en el campo de entrada de texto y seleccione Delete (Eliminar).

## Afinar las secciones de recursos y condiciones de las políticas

Puede restringir el ámbito de los permisos de un usuario mediante la especificación de recursos y condiciones en una política de AWS Identity and Access Management (IAM). Todas las acciones en una política admiten una combinación de tipos de recursos y condiciones que varía en función del comportamiento de la acción.

Cada instrucción de una política de IAM concede permiso para realizar una acción en un recurso. Cuando la acción no actúa sobre un recurso designado, o cuando se concede permiso para realizar la acción en todos los recursos, el valor del recurso en la política es un comodín (\*). Para muchas acciones, puede restringir los recursos que un usuario puede modificar si especifica el nombre de recurso de Amazon (ARN) de un recurso o un patrón de ARN que coincida con varios recursos.

Para restringir los permisos por recurso, especifique el recurso por ARN.

### Formato de ARN de recurso de Lambda

- Función: `arn:aws:lambda:us-west-2:123456789012:function:my-function`
- Versión de la función: `arn:aws:lambda:us-west-2:123456789012:function:my-function:1`
- Alias de la función: `arn:aws:lambda:us-west-2:123456789012:function:my-function:TEST`
- Asignación de origen de eventos: `arn:aws:lambda:us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47`
- Capa: `arn:aws:lambda:us-west-2:123456789012:layer:my-layer`

- Versión de la capa: `arn:aws:lambda:us-west-2:123456789012:layer:my-layer:1`

Por ejemplo, la siguiente política permite que un usuario en una Cuenta de AWS 123456789012 invoque una función denominada `my-function` en la región de AWS Oeste de EE. UU. (Oregón).

Example invocar política de función

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Invoke",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
    }
  ]
}
```

Se trata de un caso especial donde el identificador de la acción (`lambda:InvokeFunction`) difiere de la operación de la API ([Invoke](#)). Para otras acciones, el identificador de la acción es el nombre de la operación con el prefijo `lambda:`.

## Secciones

- [Comprensión de la sección de condiciones de las políticas](#)
- [Hacer referencia a las funciones en la sección de recursos de las políticas](#)
- [Acciones de función admitidas](#)
- [Acciones de asignación de orígenes de eventos admitidos](#)
- [Acciones de capas admitidas](#)

## Comprensión de la sección de condiciones de las políticas

Las condiciones son un elemento opcional de la política que aplica lógica adicional para determinar si se permite o no una acción. Además de las [condiciones](#) comunes que admiten todas las acciones, Lambda define tipos de condiciones que puede utilizar para restringir los valores de parámetros adicionales en algunas acciones.

Por ejemplo, la condición `lambda:Principal` permite restringir el servicio o la cuenta a los que un usuario puede conceder acceso de invocación en la [política basada en recursos](#) de una función. La siguiente política permite a un usuario conceder permiso a los temas de Amazon Simple Notification Service (Amazon SNS) para invocar una función denominada `test`.

Example administrar permisos de una política de función

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageFunctionPolicy",
      "Effect": "Allow",
      "Action": [
        "lambda:AddPermission",
        "lambda:RemovePermission"
      ],
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:test:*",
      "Condition": {
        "StringEquals": {
          "lambda:Principal": "sns.amazonaws.com"
        }
      }
    }
  ]
}
```

La condición requiere que el principal sea Amazon SNS y no otro servicio o cuenta. El patrón de recursos requiere que el nombre de la función sea `test` e incluye un número de versión o alias. Por ejemplo, `test:v1`.

Para obtener más información sobre los recursos y las condiciones para Lambda y otros servicios de AWS, consulte [Acciones, recursos y claves de condición de los servicios de AWS](#) en la Referencia de autorización de servicios.

## Hacer referencia a las funciones en la sección de recursos de las políticas

Puede hacer referencia a una función de Lambda en la instrucción de una política con un nombre de recurso de Amazon (ARN). El formato de un ARN de función depende de si se hace referencia a toda la función (incompleto), o a la [versión](#) o el [alias](#) de la función (completo).



Cuando realizan llamadas a la API de Lambda, los usuarios pueden especificar una versión o un alias al pasar el ARN de la versión o el alias en el parámetro [GetFunction](#) de `FunctionName`, o al establecer un valor en el parámetro [GetFunction](#) de `Qualifier`. Lambda toma decisiones de autorización comparando el elemento de recurso de la política de IAM con el `FunctionName` y el `Qualifier` pasado en las llamadas a la API. Si hay un error de coincidencia, Lambda deniega la solicitud.

Tanto si permite o deniega una acción en la función, debe utilizar los tipos de ARN de función correctos en la instrucción de la política para obtener los resultados esperados. Por ejemplo, si su política hace referencia al ARN incompleto, Lambda acepta las solicitudes que hacen referencia al ARN incompleto, pero deniega las solicitudes que hacen referencia a un ARN completo.

#### Note

No se puede utilizar un carácter comodín (\*) para que coincida con el ID de cuenta. Para obtener más información sobre la sintaxis aceptada, consulte la [referencia de la política JSON de IAM](#) en la Guía del usuario de IAM.

#### Example permitir la invocación de un ARN incompleto

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction"
    }
  ]
}
```

Si su política hace referencia a un ARN completo específico, Lambda acepta las solicitudes que hacen referencia al ARN, pero deniega las solicitudes que hagan referencia a un ARN incompleto o a otro ARN completo, como `myFunction:2`.

#### Example permitir la invocación de un ARN completo específico

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": "lambda:InvokeFunction",
    "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:1"
  }
]
}

```

Si su política hace referencia a cualquier ARN completo mediante `*`, Lambda acepta cualquier ARN completo, pero deniega las solicitudes que hagan referencia a un ARN incompleto.

Example permitir la invocación de cualquier ARN completo

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction:*"
    }
  ]
}

```

Si su política hace referencia a cualquier ARN mediante `*`, Lambda acepta cualquier ARN completo o incompleto.

Example permitir la invocación de cualquier ARN completo o incompleto

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:myFunction*"
    }
  ]
}

```

## Acciones de función admitidas

Las acciones que operan sobre una función se pueden restringir a una función específica por ARN de función, versión o alias, tal y como se describe en la siguiente tabla. Las acciones que no admiten restricciones de recursos se conceden para todos los recursos (\*).

Acción	Recurso	Condición
<a href="#">AddPermission</a>	Función	lambda:Principal
<a href="#">RemovePermission</a>	Versión de función	aws:ResourceTag/\${TagKey}
	Alias de función	lambda:FunctionUrl AuthType
<a href="#">Invoke</a>  Permiso: lambda:InvokeFunction	Función	aws:ResourceTag/\${TagKey}
	Versión de función	lambda:EventSourceToken
	Alias de función	
<a href="#">CreateFunction</a>	Función	lambda:CodeSigningConfigArn
		lambda:Layer
		lambda:VpcIds
		lambda:SubnetIds
		lambda:SecurityGroupIds
		aws:ResourceTag/\${TagKey}
		aws:RequestTag/\${TagKey}
aws:TagKeys		
<a href="#">UpdateFunctionConfiguration</a>	Función	lambda:CodeSigningConfigArn

Acción	Recurso	Condición
		<code>lambda:Layer</code> <code>lambda:VpcIds</code> <code>lambda:SubnetIds</code> <code>lambda:SecurityGroupIds</code> <code>aws:ResourceTag/\${TagKey}</code>

Acción	Recurso	Condición
<a href="#">CreateAlias</a>	Función	aws:ResourceTag/\${TagKey}
<a href="#">DeleteAlias</a>		
<a href="#">DeleteFunction</a>		
<a href="#">DeleteFunctionCodeSigningConfig</a>		
<a href="#">DeleteFunctionConcurrency</a>		
<a href="#">GetAlias</a>		
<a href="#">GetFunction</a>		
<a href="#">GetFunctionCodeSigningConfig</a>		
<a href="#">GetFunctionConcurrency</a>		
<a href="#">GetFunctionConfiguration</a>		
<a href="#">GetPolicy</a>		
<a href="#">ListProvisionedConcurrencyConfigs</a>		
<a href="#">ListAliases</a>		
<a href="#">ListTags</a>		
<a href="#">ListVersionsByFunction</a>		
<a href="#">PublishVersion</a>		
<a href="#">PutFunctionCodeSigningConfig</a>		
<a href="#">PutFunctionConcurrency</a>		
<a href="#">UpdateAlias</a>		
<a href="#">UpdateFunctionCode</a>		

Acción	Recurso	Condición
<a href="#">CreateFunctionUrlConfig</a>	Función	lambda:FunctionUrl
<a href="#">DeleteFunctionUrlConfig</a>	Alias de función	AuthType
<a href="#">GetFunctionUrlConfig</a>		aws:ResourceTag/\${TagKey}
<a href="#">UpdateFunctionUrlConfig</a>		
<a href="#">ListFunctionUrlConfigs</a>		lambda:FunctionUrl AuthType
<a href="#">DeleteFunctionEventInvokeConfig</a>	Función	aws:ResourceTag/\${TagKey}
<a href="#">GetFunctionEventInvokeConfig</a>		
<a href="#">ListFunctionEventInvokeConfigs</a>		
<a href="#">PutFunctionEventInvokeConfig</a>		
<a href="#">UpdateFunctionEventInvokeConfig</a>		
<a href="#">DeleteProvisionedConcurrencyConfig</a>	Alias de función	aws:ResourceTag/\${TagKey}
<a href="#">GetProvisionedConcurrencyConfig</a>	Versión de función	
<a href="#">PutProvisionedConcurrencyConfig</a>		
<a href="#">GetAccountSettings</a>	*	Ninguna
<a href="#">ListFunctions</a>		
<a href="#">TagResource</a>	Función	aws:ResourceTag/\${TagKey} aws:RequestTag/\${TagKey} aws:TagKeys
<a href="#">UntagResource</a>	Función	aws:ResourceTag/\${TagKey} aws:TagKeys

## Acciones de asignación de orígenes de eventos admitidos

Para las [asignaciones de orígenes de eventos](#), puede restringir los permisos de eliminación y actualización a un origen de eventos específico. La condición `lambda:FunctionArn` le permite restringir las funciones que un usuario puede configurar para invocar un origen de eventos.

Para estas acciones, el recurso es el mapeo de origen de eventos, por lo que Lambda proporciona una condición que permite restringir el permiso según la función que invoca el mapeo de fuente de eventos.

### Acciones de asignación de origen de eventos

Acción	Recurso	Condición
<a href="#">DeleteEventSourceMapping</a> <a href="#">UpdateEventSourceMapping</a>	Asignación de orígenes de eventos	<code>lambda:FunctionArn</code>
<a href="#">CreateEventSourceMapping</a> <a href="#">GetEventSourceMapping</a>	*	<code>lambda:FunctionArn</code>
<a href="#">ListEventSourceMappings</a>	*	Ninguna

## Acciones de capas admitidas

Las acciones de capa permiten restringir las capas que un usuario puede administrar o utilizar con una función. Las acciones relacionadas con el uso y los permisos de capa actúan sobre una versión de una capa, mientras que `PublishLayerVersion` actúa sobre un nombre de capa. Pueden utilizar cualquiera de ellas con comodines para restringir las capas que un usuario puede utilizar por nombre.

### Note

La acción [GetLayerVersion](#) también explica lo siguiente: [GetLayerVersionByArn](#). Lambda no es compatible con `GetLayerVersionByArn` como una acción de IAM.

## Acciones de capa

Acción	Recurso	Condición
<a href="#">AddLayerVersionPermission</a>	Versión de la capa	Ninguna
<a href="#">RemoveLayerVersionPermission</a>		
<a href="#">GetLayerVersion</a>		
<a href="#">GetLayerVersionPolicy</a>		
<a href="#">DeleteLayerVersion</a>		
<a href="#">ListLayerVersions</a>	Capa	Ninguna
<a href="#">PublishLayerVersion</a>		
<a href="#">ListLayers</a>	*	Ninguna



# Seguridad en AWS Lambda

La seguridad en la nube de AWS es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y el usuario. El [modelo de responsabilidad compartida](#) la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta los servicios de AWS en la nube de AWS. AWS también proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los [programas de conformidad de AWS](#). Para obtener más información sobre los programas de conformidad que se aplican a AWS Lambda, consulte [Servicios de AWS en el ámbito del programa de conformidad](#).
- Seguridad en la nube: su responsabilidad se determina según el servicio de AWS que utilice. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y la normativa aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza Lambda. En los siguientes temas, se le mostrará cómo configurar Lambda para satisfacer sus objetivos de seguridad y conformidad. También puede aprender a utilizar otros servicios de AWS que lo ayuden a monitorear y proteger los recursos de Lambda.

Para obtener más información acerca de cómo aplicar los principios de seguridad a las aplicaciones de Lambda, consulte [Security](#) en Serverless Land.

## Temas

- [Protección de los datos en AWS Lambda](#)
- [Administración de identidades y accesos para AWS Lambda](#)
- [Cree una estrategia de gobernanza para las funciones y capas de Lambda](#)
- [Validación de conformidad en AWS Lambda](#)
- [Resiliencia en AWS Lambda](#)
- [Seguridad de la infraestructura en AWS Lambda](#)
- [Uso de la firma de código para verificar la integridad del código con Lambda](#)

# Protección de los datos en AWS Lambda

El [modelo de responsabilidad compartida](#) de AWS se aplica a la protección de datos de AWS Lambda. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global que ejecuta toda la Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las [Preguntas frecuentes sobre la privacidad de datos](#). Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el [Modelo de responsabilidad compartida de AWS y GDPR](#) en el Blog de seguridad de AWS.

Con fines de protección de datos, recomendamos proteger las credenciales de la Cuenta de AWS y configurar cuentas de usuario individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice la autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos de AWS. Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure los registros de API y de actividad de los usuarios con AWS CloudTrail.
- Utilice las soluciones de cifrado de AWS, junto con todos los controles de seguridad predeterminados dentro de los servicios de Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados FIPS 140-3 al acceder a AWS a través de una interfaz de línea de comandos o una API, utilice un punto de conexión de FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte [Estándar de procesamiento de la información federal \(FIPS\) 140-3](#).

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Incluye las situaciones en las que debe trabajar con Lambda u otros Servicios de AWS a través de la consola, la API, la AWS CLI o los SDK de AWS. Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los

registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

## Secciones

- [Cifrado en tránsito](#)
- [Cifrado en reposo](#)

## Cifrado en tránsito

Los puntos de enlace de API de Lambda solo admiten conexiones seguras a través de HTTPS. Al administrar recursos de Lambda con la AWS Management Console, el AWS SDK o la API de Lambda, todas las comunicaciones se cifran con Transport Layer Security (TLS). Para obtener una lista completa de puntos de enlace de API, consulte [Regiones y puntos de enlace de AWS](#) en la Referencia general de AWS.

Cuando [conecta su función a un sistema de archivos](#), Lambda utiliza el cifrado en tránsito para todas las conexiones. Para obtener más información, consulte [Cifrado de datos en Amazon EFS](#) en la Guía del usuario de Amazon Elastic File System.

Cuando utilice [variables de entorno](#), puede permitir que las funciones auxiliares de cifrado de la consola utilicen el cifrado del lado del cliente para proteger las variables de entorno en tránsito. Para obtener más información, consulte [Asegurar las variables de entorno Lambda](#).

## Cifrado en reposo

Lambda siempre cifra las variables de entorno en reposo. De forma predeterminada, Lambda utiliza una AWS KMS key que crea en su cuenta para cifrar las variables de entorno. Esta Clave administrada de AWS se llama `aws/lambda`.

En una base por función, puede configurar de forma opcional Lambda para utilizar una clave administrada por el cliente en lugar de la Clave administrada de AWS predeterminada para cifrar las variables de entorno. Para obtener más información, consulte [Asegurar las variables de entorno Lambda](#).

Lambda siempre cifra los archivos que carga en Lambda, incluidos los [paquetes de implementación](#) y los [archivos de capas](#).

Amazon CloudWatch Logs y AWS X-Ray también cifran los datos de manera predeterminada y se pueden configurar para utilizar una clave administrada por el cliente. Para obtener más información, consulte [Cifrar datos de registro en CloudWatch Logs](#) y [Protección de los datos en AWS X-Ray](#).

## Administración de identidades y accesos para AWS Lambda

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda a los administradores a controlar de forma segura el acceso a los recursos de AWS. Los administradores de IAM controlan quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos de Lambda. IAM es un Servicio de AWS que se puede utilizar sin cargo adicional.

### Temas

- [Público](#)
- [Autenticación con identidades](#)
- [Administración de acceso mediante políticas](#)
- [Cómo AWS Lambda funciona con IAM](#)
- [Ejemplos de políticas basadas en identidades de AWS Lambda](#)
- [Políticas administradas de AWS para AWS Lambda](#)
- [Solución de problemas de identidades de AWS Lambda y accesos](#)

## Público

La forma en que utilice AWS Identity and Access Management (IAM) difiere, en función del trabajo que realice en Lambda.

Usuario de servicio: si utiliza el servicio de Lambda para realizar su trabajo, su administrador le proporciona las credenciales y los permisos que necesita. A medida que utilice más características de Lambda para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en Lambda, consulte [Solución de problemas de identidades de AWS Lambda y accesos](#).

Administrador de servicio: si está a cargo de los recursos de Lambda en su empresa, probablemente tenga acceso completo a Lambda. Su trabajo consiste en determinar a qué características y

recursos de Lambda deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar IAM con Lambda, consulte [Cómo AWS Lambda funciona con IAM](#).

Administrador de IAM: si es un administrador de IAM, es posible que desee obtener información sobre cómo escribir políticas para administrar el acceso a Lambda. Para consultar ejemplos de políticas basadas en la identidad de Lambda que puede utilizar en IAM, consulte [Ejemplos de políticas basadas en identidades de AWS Lambda](#).

## Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe estar autenticado (haber iniciado sesión en AWS) como Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Puede iniciar sesión en AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (del Centro de identidades de IAM), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accede a AWS mediante la federación, está asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en AWS Management Console o en el portal de acceso AWS. Para obtener más información sobre el inicio de sesión en AWS, consulte [Cómo iniciar sesión en su Cuenta de AWS](#) en la Guía del usuario de AWS Sign-In.

Si accede a AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de la línea de comandos (CLI) para firmar criptográficamente las solicitudes mediante el uso de las credenciales. Si no usa las herramientas de AWS, debe firmar las solicitudes. Para obtener más información sobre la firma de solicitudes, consulte [Firma de solicitudes API de AWS](#) en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, AWS le recomienda el uso de la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte [Autenticación multifactor](#) en la Guía del usuario de AWS IAM Identity Center y [Uso de la autenticación multifactor \(MFA\) en AWS](#) en la Guía del usuario de IAM.

## Usuario raíz de Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión que tiene acceso completo a todos los recursos y Servicios de AWS de la cuenta. Esta identidad recibe el nombre de usuario raíz de la Cuenta de AWS y se accede a ella iniciando sesión con el email y la contraseña que utilizó para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

## Identidad federada

Como práctica recomendada, solicite que los usuarios humanos, incluidos los que requieren acceso de administrador, utilicen la federación con un proveedor de identidades para acceder a los Servicios de AWS utilizando credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidad web, el AWS Directory Service, el directorio del Identity Center, o cualquier usuario que acceda a Servicios de AWS utilizando credenciales proporcionadas a través de una fuente de identidad. Cuando identidades federadas acceden a Cuentas de AWS, asumen roles y los roles proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utilice AWS IAM Identity Center. Puede crear usuarios y grupos en el IAM Identity Center o puede conectarse y sincronizar con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus aplicaciones y Cuentas de AWS. Para obtener más información, consulte [¿Qué es el Centro de identidades de IAM?](#) en la Guía del usuario de AWS IAM Identity Center.

## Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad de la Cuenta de AWS que dispone de permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte [Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales. Para más información, consulte [Cuándo crear un usuario de IAM \(en lugar de un rol\)](#) en la Guía del usuario de IAM.

## Roles de IAM

Un [rol de IAM](#) es una identidad de la Cuenta de AWS que dispone de permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente un rol de IAM en la AWS Management Console [cambiando de roles](#). Puede asumir un rol llamando a una operación de la AWS CLI o de la API de AWS, o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte [Uso de roles de IAM](#) en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- **Acceso de usuario federado:** para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte [Creación de un rol para un proveedor de identidades de terceros](#) en la Guía del usuario de IAM. Si utiliza IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte [Conjuntos de permisos](#) en la Guía del usuario de AWS IAM Identity Center.
- **Permisos de usuario de IAM temporales:** un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- **Acceso entre cuentas:** puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. No obstante, con algunos Servicios de AWS se puede adjuntar una política directamente a un recurso (en lugar de utilizar un rol como representante). Para



obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

- **Acceso entre servicios:** algunos Servicios de AWS utilizan características de otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado a servicios.
- **Reenviar sesiones de acceso (FAS):** cuando utiliza un rol o un usuario de IAM para llevar a cabo acciones en AWS, se le considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte [Reenviar sesiones de acceso](#).
- **Rol de servicio:** un rol de servicio es un [rol de IAM](#) que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte [Creación de un rol para delegar permisos a un Servicio de AWS](#) en la Guía del usuario de IAM.
- **Rol vinculado a los servicios:** un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.
- **Aplicaciones que se ejecutan en Amazon EC2:** puede utilizar un rol de IAM que le permita administrar credenciales temporales para las aplicaciones que se ejecutan en una instancia de EC2 y realizan solicitudes a la AWS CLI o a la API de AWS. Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia de EC2. Para asignar un rol de AWS a una instancia de EC2 y ponerla a disposición de todas las aplicaciones, cree un perfil de instancia adjuntado a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia de EC2 obtener credenciales temporales. Para más información, consulte [Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2](#) en la Guía del usuario de IAM.



Para obtener información sobre el uso de los roles de IAM, consulte [Cuándo crear un rol de IAM \(en lugar de un usuario\)](#) en la Guía del usuario de IAM.

## Administración de acceso mediante políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política es un objeto de AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal (sesión de rol, usuario o usuario raíz) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte [Información general de políticas JSON](#) en la Guía del usuario de IAM.

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con dicha política puede obtener información del usuario de la AWS Management Console, la AWS CLI o la API de AWS.

### Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte [Creación de políticas de IAM](#) en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS. Las políticas administradas incluyen las políticas administradas de AWS y las políticas administradas por el cliente. Para más información sobre cómo elegir una

política administrada o una política insertada, consulte [Elegir entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

## Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe [especificar una entidad principal](#) en una política en función de recursos. Las entidades principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No se puede utilizar políticas de IAM administradas de AWS en una política basada en recursos.

## Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3, AWS WAF y Amazon VPC son ejemplos de servicios que admiten las ACL. Para obtener más información sobre las ACL, consulte [Información general de Lista de control de acceso \(ACL\)](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

## Otros tipos de políticas

AWS admite otros tipos de políticas adicionales menos frecuentes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- **Límites de permisos:** un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites

de los permisos, consulte [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.

- **Políticas de control de servicio (SCP):** las SCP son políticas de JSON que especifican los permisos máximos de una organización o una unidad organizativa en AWS Organizations. AWS Organizations es un servicio que le permite agrupar y administrar de manera centralizada varias Cuentas de AWS que posea su empresa. Si habilita todas las características en una empresa, entonces podrá aplicar políticas de control de servicio (SCP) a una o todas sus cuentas. Una SCP limita los permisos para las entidades de las cuentas de miembros, incluido cada Usuario raíz de la cuenta de AWS. Para obtener más información acerca de SCP y Organizations, consulte [Políticas de control de servicios](#) en la Guía del usuario de AWS Organizations.
- **Políticas de sesión:** las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte [Políticas de sesión](#) en la Guía del usuario de IAM.

## Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información acerca de cómo AWS decide si permitir o no una solicitud cuando hay varios tipos de políticas implicados, consulte [Lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

## Cómo AWS Lambda funciona con IAM

Antes de utilizar IAM para administrar el acceso a Lambda, conozca qué características de IAM se pueden utilizar con Lambda.

Característica de IAM	Soporte de Lambda
<a href="#">Políticas basadas en identidades</a>	Sí
<a href="#">Políticas basadas en recursos</a>	Sí
<a href="#">Acciones de políticas</a>	Sí

Característica de IAM	Soporte de Lambda
<a href="#">Recursos de políticas</a>	Sí
<a href="#">Claves de condición de política (específicas del servicio)</a>	Sí
<a href="#">ACL</a>	No
<a href="#">ABAC (etiquetas en políticas)</a>	Parcial
<a href="#">Credenciales temporales</a>	Sí
<a href="#">Sesiones de acceso directo (FAS)</a>	No
<a href="#">Roles de servicio</a>	Sí
<a href="#">Roles vinculados al servicio</a>	Parcial

Para obtener una perspectiva general sobre cómo funcionan Lambda y otros servicios de AWS con la mayoría de las características de IAM, consulte [Servicios de AWS que funcionan con IAM](#) en la Guía del usuario de IAM.

## Políticas basadas en identidad para Lambda

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte [Creación de políticas de IAM](#) en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está adjunto. Para más información sobre los elementos que puede utilizar en una política de JSON, consulte [Referencia de los elementos de las políticas de JSON de IAM](#) en la Guía del usuario de IAM.

## Ejemplos de políticas basadas en identidad para Lambda

Para ver ejemplos de políticas basadas en identidad de Lambda, consulte [Ejemplos de políticas basadas en identidades de AWS Lambda](#).

## Políticas basadas en recursos de Lambda

Compatibilidad con las políticas basadas en recursos: sí

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe [especificar una entidad principal](#) en una política en función de recursos. Las entidades principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Para habilitar el acceso entre cuentas, puede especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando la entidad principal y el recurso se encuentran en Cuentas de AWS diferentes, un administrador de IAM de la cuenta de confianza también debe conceder a la entidad principal (usuario o rol) permiso para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política en función de recursos concede el acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para más información, consulte [Cross account resource access in IAM](#) en la Guía del usuario de IAM.

Puede asociar una política basada en recursos a una capa o función de Lambda. Esta política define qué entidades principales pueden llevar a cabo acciones en la capa o en la función.

Para obtener información sobre cómo asociar una política basada en recursos a una capa o función, consulte [Trabajar con políticas de IAM basadas en recursos en Lambda](#).

## Acciones de políticas de Lambda

Compatibilidad con las acciones de política: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones de la política generalmente tienen el mismo nombre que la operación de API de AWS asociada. Hay algunas excepciones, como acciones de solo permiso que no tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de las acciones de Lambda, consulte [Acciones definidas por AWS Lambda](#) en la Referencia de autorizaciones de servicio.

Las acciones de políticas de Lambda utilizan el siguiente prefijo antes de la acción:

```
lambda
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [  
  "lambda:action1",  
  "lambda:action2"  
]
```

Para ver ejemplos de políticas basadas en identidad de Lambda, consulte [Ejemplos de políticas basadas en identidades de AWS Lambda](#).

## Recursos de políticas de Lambda

Compatibilidad con los recursos de políticas: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el [Nombre de recurso de Amazon \(ARN\)](#). Puede

hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (\*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de los tipos de recursos de Lambda y sus ARN, consulte [Tipos de recursos definidos por AWS Lambda](#) en la Referencia de autorizaciones de servicio. Para obtener información sobre las acciones con las que puede especificar el ARN de cada recurso, consulte [Acciones definidas por AWS Lambda](#).

Para ver ejemplos de políticas basadas en identidad de Lambda, consulte [Ejemplos de políticas basadas en identidades de AWS Lambda](#).

## Claves de condición de política para Lambda

Compatibilidad con claves de condición de políticas específicas del servicio: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen [operadores de condición](#), tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación AND lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición con una operación lógica OR. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado con su nombre de usuario de IAM. Para más información, consulte [Elementos de la política de IAM: variables y etiquetas](#) en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición globales de AWS, consulte [Claves de contexto de condición globales de AWS](#) en la Guía del usuario de IAM.

Para ver una lista de las claves de condición de Lambda, consulte [Claves de condición para AWS Lambda](#) en la Referencia de autorizaciones de servicio. Para obtener más información sobre las acciones y los recursos con los que puede utilizar una clave de condición, consulte [Acciones definidas por AWS Lambda](#).

Para ver ejemplos de políticas basadas en identidad de Lambda, consulte [Ejemplos de políticas basadas en identidades de AWS Lambda](#).

## ACL en Lambda

Compatibilidad con ACL: no

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

## ABAC con Lambda

Compatibilidad con ABAC (etiquetas en las políticas): parcial

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos. En AWS, estos atributos se denominan etiquetas. Puede adjuntar etiquetas a entidades de IAM (usuarios o roles) y a muchos recursos de AWS. El etiquetado de entidades y recursos es el primer paso de ABAC. A continuación, designa las políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso al que se intenta acceder.

ABAC es útil en entornos que crecen con rapidez y ayuda en situaciones en las que la administración de las políticas resulta engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el [elemento de condición](#) de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.



Para obtener más información sobre ABAC, consulte [¿Qué es ABAC?](#) en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte [Uso del control de acceso basado en atributos \(ABAC\)](#) en la Guía del usuario de IAM.

Para obtener más información acerca del etiquetado de recursos de Lambda, consulte [Control de acceso basado en atributos para Lambda](#).

## Uso de credenciales temporales con Lambda

Compatibilidad con credenciales temporales: sí

Algunos Servicios de AWS no funcionan cuando inicia sesión con credenciales temporales. Para obtener información adicional, incluida la información sobre qué Servicios de AWS funcionan con credenciales temporales, consulte [Servicios de AWS que funcionan con IAM](#) en la Guía del usuario de IAM.

Utiliza credenciales temporales si inicia sesión en la AWS Management Console con cualquier método, excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accede a AWS utilizando el enlace de inicio de sesión único (SSO) de la empresa, ese proceso crea automáticamente credenciales temporales. También crea automáticamente credenciales temporales cuando inicia sesión en la consola como usuario y luego cambia de rol. Para más información sobre el cambio de roles, consulte [Cambio a un rol \(consola\)](#) en la Guía del usuario de IAM.

Puede crear credenciales temporales de forma manual mediante la AWS CLI o la API de AWS. A continuación, puede usar esas credenciales temporales para acceder a AWS. AWS recomienda generar credenciales temporales de forma dinámica en lugar de usar claves de acceso a largo plazo. Para más información, consulte [Credenciales de seguridad temporales en IAM](#).

## Sesiones de acceso directo para Lambda

Compatibilidad con las sesiones de acceso directo (FAS): no

Cuando utiliza un usuario o un rol de IAM para llevar a cabo acciones en AWS, se lo considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte [Reenviar sesiones de acceso](#).

## Roles de servicio de Lambda

Compatibilidad con roles de servicio: sí

Un rol de servicio es un [rol de IAM](#) que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte [Creación de un rol para delegar permisos a un Servicio de AWS](#) en la Guía del usuario de IAM.

En Lambda, un rol de servicio se conoce como un rol de [ejecución](#).

### Warning

Cambiar los permisos de un rol de ejecución podría interrumpir la funcionalidad de Lambda.

## Roles vinculados a servicios para Lambda

Compatibilidad con roles vinculados al servicio: parcial

Un rol vinculado al servicio es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Lambda no tiene roles vinculados a servicios, pero Lambda@Edge sí. Para obtener más información, consulte [Uso de roles vinculados a servicios de en la Guía para desarrolladores de Lambda@Edge](#) en la Guía para desarrolladores de Amazon CloudFront.

Para más información sobre cómo crear o administrar roles vinculados a servicios, consulte [Servicios de AWS que funcionan con IAM](#). Busque un servicio en la tabla que incluya Yes en la columna Rol vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

## Ejemplos de políticas basadas en identidades de AWS Lambda

De forma predeterminada, los usuarios y roles no tienen permiso para crear ni modificar los recursos de Lambda. Tampoco pueden realizar tareas mediante la AWS Management Console, la AWS Command Line Interface (AWS CLI) o la API de AWS. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan.

A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte [Creación de políticas de IAM](#) en la Guía del usuario de IAM.

A fin de obtener más información sobre las acciones y los tipos de recursos definidos por Lambda, incluido el formato de los ARN para cada tipo de recurso, consulte [Acciones, recursos y claves de condición para AWS Lambda](#) en la Referencia de autorizaciones de servicio.

## Temas

- [Prácticas recomendadas sobre las políticas](#)
- [Uso de la consola de Lambda](#)
- [Cómo permitir a los usuarios consultar sus propios permisos](#)

## Prácticas recomendadas sobre las políticas

Las políticas basadas en identidades determinan si alguien puede crear, acceder o eliminar los recursos de Lambda de la cuenta. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas administradas por AWS y continúe con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de trabajo, utilice las políticas administradas por AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas administradas por el cliente de AWS específicas para sus casos de uso a fin de reducir aún más los permisos. Con el fin de obtener más información, consulte las [políticas administradas por AWS](#) o las [políticas administradas por AWS para funciones de trabajo](#) en la Guía de usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte [Políticas y permisos en IAM](#) en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo,

puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puede usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado como, por ejemplo, AWS CloudFormation. Para obtener más información, consulte [Elementos de la política de JSON de IAM: Condición](#) en la Guía del usuario de IAM.

- Utilice el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte [Política de validación de Analizador de acceso de IAM](#) en la Guía de usuario de IAM.
- Solicite la autenticación multifactor (MFA): si se encuentra en una situación en la que necesita usuarios raíz o de IAM en su Cuenta de AWS, active la MFA para mayor seguridad. Para solicitar la MFA cuando se invocan las operaciones de la API, agregue las condiciones de la MFA a sus políticas. Para más información, consulte [Configuración del acceso a una API protegido por MFA](#) en la Guía de usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte las [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.

## Uso de la consola de Lambda

Para acceder a la consola de AWS Lambda, debe tener un conjunto mínimo de permisos. Estos permisos deben permitir que registre y consulte los detalles acerca de los recursos de Lambda en su cuenta de Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario que conceda permisos mínimos para la consola a los usuarios que solo realizan llamadas a la AWS CLI o a la API de AWS. En su lugar, permite acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para obtener un ejemplo de política que concede un acceso mínimo para el desarrollo de funciones, consulte [Concesión de acceso a una función de Lambda a los usuarios](#). Además de las API de Lambda, la consola de Lambda utiliza otros servicios para mostrar la configuración de activación y permitir agregar nuevos desencadenadores. Si sus usuarios usan Lambda con otros servicios,

también necesitan acceso a esos servicios. Para obtener más información sobre cómo configurar otros servicios con Lambda, consulte [Invocar Lambda con eventos de otros servicios de AWS](#).

## Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para llevar a cabo esta acción en la consola o mediante programación con la AWS CLI o la API de AWS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

## Políticas administradas de AWS para AWS Lambda

Una política administrada de AWS es una política independiente que AWS crea y administra. Las políticas administradas de AWS se diseñan para ofrecer permisos para muchos casos de uso comunes, por lo que puede empezar a asignar permisos a los usuarios, grupos y roles.

Considere que es posible que las políticas administradas por AWS no concedan permisos de privilegio mínimo para los casos de uso concretos, ya que están disponibles para que las utilicen todos los clientes de AWS. Se recomienda definir [políticas administradas por el cliente](#) para los casos de uso a fin de reducir aún más los permisos.

No puede cambiar los permisos definidos en las políticas administradas de AWS. Si AWS actualiza los permisos definidos en una política administrada de AWS, la actualización afecta a todas las identidades de entidades principales (usuarios, grupos y roles) a las que está adjunta la política. Lo más probable es que AWS actualice una política administrada de AWS cuando se lance un nuevo Servicio de AWS o las operaciones de la API nuevas estén disponibles para los servicios existentes.

Para obtener más información, consulte [Políticas administradas por AWS](#) en la Guía del usuario de IAM.

### Temas

- [AWSpolítica gestionada: AWSLambda\\_FullAccess](#)
- [AWSpolítica gestionada: AWSLambda\\_ReadOnlyAccess](#)
- [AWSpolítica gestionada: AWSLambdaBasicExecutionRole](#)
- [AWSpolítica gestionada: AWSLambdaDynamoDBExecutionRole](#)
- [AWSpolítica gestionada: AWSLambdaENIManagementAccess](#)
- [AWSpolítica gestionada: AWSLambdaExecute](#)
- [AWSpolítica gestionada: AWSLambdaInvocation -DynamoDB](#)
- [AWSpolítica gestionada: AWSLambdaKinesisExecutionRole](#)
- [AWSpolítica gestionada: AWSLambdaMSKExecutionRole](#)
- [AWSpolítica gestionada: AWSLambdaRole](#)
- [AWSpolítica gestionada: AWSLambdaSQSQueueExecutionRole](#)
- [AWSpolítica gestionada: AWSLambdaVPCLambdaAccessExecutionRole](#)
- [Actualizaciones de Lambda en las políticas administradas de AWS](#)

## AWSpolítica gestionada: AWSLambda\_FullAccess

Esta política concede acceso completo a las acciones de Lambda. También concede permisos a otros servicios de AWS que se utilizan para desarrollar y mantener los recursos de Lambda.

Puede asociar la política `AWSLambda_FullAccess` a sus usuarios, grupos y roles.

### Detalles de los permisos

Esta política incluye los permisos siguientes:

- `lambda`: permite a las entidades principales obtener acceso completo a Lambda.
- `cloudformation`: permite a las entidades principales describir las pilas de AWS CloudFormation y enumerar los recursos de esas pilas.
- `cloudwatch`— Permite a los directores enumerar CloudWatch las métricas de Amazon y obtener datos de las métricas.
- `ec2`: permite a las entidades principales describir los grupos de seguridad, las subredes y las VPC.
- `iam`: permite a las entidades principales obtener las políticas, las versiones de las políticas, los roles, las políticas de roles, las políticas de roles asociadas y la lista de roles. Esta política también permite a las entidades principales transferir roles a Lambda. El permiso `PassRole` se usa al asignar un rol de ejecución a una función.
- `kms`: permite a las entidades principales enumerar los alias.
- `logs`— Permite a los directores describir los grupos de CloudWatch registros de Amazon. En el caso de los grupos de registro que están asociados a una función de Lambda, esta política permite que la entidad principal describa los flujos de registro, obtenga los eventos de registro y filtre los eventos de registro.
- `states`: permite a las entidades principales describir y enumerar las máquinas de estado de AWS Step Functions.
- `tag`: permite a las entidades principales obtener recursos en función de sus etiquetas.
- `xray`: permite a las entidades principales obtener resúmenes de registros de seguimiento de AWS X-Ray y recuperar una lista de los registros de seguimiento especificados por ID.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambda\\_FullAccess](#) la Guía de referencia de políticas AWS administradas.

## AWSpolítica gestionada: AWSLambda\_ReadOnlyAccess

Esta política concede acceso de solo lectura a los recursos de Lambda y a otros servicios de AWS que se utilizan para desarrollar y mantener los recursos de Lambda.

Puede asociar la política AWSLambda\_ReadOnlyAccess a sus usuarios, grupos y roles.

### Detalles de los permisos

Esta política incluye los permisos siguientes:

- `lambda`: permite a las entidades principales obtener y enumerar todos los recursos.
- `cloudformation`: permite a las entidades principales describir y enumerar las pilas de AWS CloudFormation y enumerar los recursos de esas pilas.
- `cloudwatch`— Permite a los directores enumerar CloudWatch las métricas de Amazon y obtener datos de las métricas.
- `ec2`: permite a las entidades principales describir los grupos de seguridad, las subredes y las VPC.
- `iam`: permite a las entidades principales obtener las políticas, las versiones de las políticas, los roles, las políticas de roles, las políticas de roles asociadas y la lista de roles.
- `kms`: permite a las entidades principales enumerar los alias.
- `logs`— Permite a los directores describir los grupos de CloudWatch registros de Amazon. En el caso de los grupos de registro que están asociados a una función de Lambda, esta política permite que la entidad principal describa los flujos de registro, obtenga los eventos de registro y filtre los eventos de registro.
- `states`: permite a las entidades principales describir y enumerar las máquinas de estado de AWS Step Functions.
- `tag`: permite a las entidades principales obtener recursos en función de sus etiquetas.
- `xray`: permite a las entidades principales obtener resúmenes de registros de seguimiento de AWS X-Ray y recuperar una lista de los registros de seguimiento especificados por ID.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambda\\_ReadOnlyAccess](#) la Guía de referencia de políticas AWS administradas.

## AWSpolítica gestionada: AWSLambdaBasicExecutionRole

Esta política otorga permisos para cargar registros en CloudWatch Logs.



Puede asociar la política `AWSLambdaBasicExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulta [AWSLambdaBasicExecutionRole](#) la Guía de referencia de políticas AWS gestionadas.

### AWSpolítica gestionada: `AWSLambdaDynamoDBExecutionRole`

Esta política concede permisos para leer registros de una transmisión de Amazon DynamoDB y escribirlos en Logs. CloudWatch

Puede asociar la política `AWSLambdaDynamoDBExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte la Guía [AWSLambdaDynamoDBExecutionRole](#) de referencia de políticas AWS administradas.

### AWSpolítica gestionada: `AWSLambdaENIManagementAccess`

Esta política concede permisos para crear, describir y eliminar las interfaces de red elásticas que utiliza una función de Lambda habilitada para VPC.

Puede asociar la política `AWSLambdaENIManagementAccess` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluidos el documento de política de JSON y las versiones de la política, consulte [AWSLambdaENIManagementAccess](#) la Guía de referencia de políticas AWS gestionadas.

### AWSpolítica gestionada: `AWSLambdaExecute`

Esta política otorga PUT GET acceso a Amazon Simple Storage Service y acceso total a CloudWatch Logs.

Puede asociar la política `AWSLambdaExecute` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambdaExecute](#) la Guía de referencia de políticas AWS administradas.

### AWSpolítica gestionada: `AWSLambdaInvocation -DynamoDB`

Esta política concede acceso de lectura a Amazon DynamoDB Streams.

Puede asociar la política `AWSLambdaInvocation-DynamoDB` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambdaInvocation-DynamoDB](#) en la Guía de referencia de políticas AWS gestionadas.

### **AWSpolítica gestionada: AWSLambdaKinesisExecutionRole**

Esta política concede permisos para leer eventos de una transmisión de datos de Amazon Kinesis y escribirlos en Logs. CloudWatch

Puede asociar la política `AWSLambdaKinesisExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambdaKinesisExecutionRole](#) la Guía de referencia de políticas AWS administradas.

### **AWSpolítica gestionada: AWSLambdaMSKExecutionRole**

Esta política concede permisos para leer y acceder a los registros de un clúster de Amazon Managed Streaming for Apache Kafka, gestionar interfaces de red elásticas y escribir CloudWatch en Logs.

Puede asociar la política `AWSLambdaMSKExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambdaMSKExecutionRole](#) la Guía de referencia de políticas AWS administradas.

### **AWSpolítica gestionada: AWSLambdaRole**

Esta política concede permisos para invocar funciones de Lambda.

Puede asociar la política `AWSLambdaRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluidos el documento de política de JSON y las versiones de la política, consulte [AWSLambdaRole](#) la Guía de referencia de políticas AWS gestionadas.

### **AWSpolítica gestionada: AWSLambdaSQSQueueExecutionRole**

Esta política concede permisos para leer y eliminar mensajes de una cola de Amazon Simple Queue Service y concede permisos de escritura a CloudWatch Logs.

Puede asociar la política `AWSLambdaSQSQueueExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte la Guía [AWSLambdaSQSQueueExecutionRole](#) de referencia de políticas AWS administradas.

### AWS política gestionada: `AWSLambdaVPCLambdaAccessExecutionRole`

Esta política concede permisos para gestionar las interfaces de red elásticas dentro de una Amazon Virtual Private Cloud y escribir en CloudWatch los registros.

Puede asociar la política `AWSLambdaVPCLambdaAccessExecutionRole` a sus usuarios, grupos y roles.

Para obtener más información sobre esta política, incluido el documento de política de JSON y las versiones de la política, consulte [AWSLambdaVPCLambdaAccessExecutionRole](#) la Guía de referencia de políticas AWS administradas.

### Actualizaciones de Lambda en las políticas administradas de AWS

Cambio	Descripción	Fecha
<a href="#">AWSLambdaVPCLambdaAccessExecutionRole</a> — Cambio	Lambda actualizó la <code>AWSLambdaVPCLambdaAccessExecutionRole</code> política para permitir la acción. <code>ec2:DescribeSubnets</code>	5 de enero de 2024
<a href="#">AWSLambda_ReadOnlyAccess</a> — Cambiar	Lambda actualizó la política <code>AWSLambda_ReadOnlyAccess</code> para permitir que las entidades principales enumeren las pilas de AWS CloudFormation.	27 de julio de 2023
AWS Lambda comenzó el seguimiento de los cambios	AWS Lambda comenzó el seguimiento de los cambios de las políticas administradas de AWS.	27 de julio de 2023

## Solución de problemas de identidades de AWS Lambda y accesos

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que es posible que surjan cuando se trabaja con Lambda e IAM.

### Temas

- [No tengo autorización para realizar una acción en Lambda](#)
- [No tengo autorización para realizar la operación iam:PassRole](#)
- [Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de Lambda](#)

### No tengo autorización para realizar una acción en Lambda

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM mateojackson intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *my-example-widget*, pero no tiene los permisos ficticios `lambda:GetWidget`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
lambda:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario mateojackson debe actualizarse para permitir el acceso al recurso *my-example-widget* mediante la acción `lambda:GetWidget`.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

### No tengo autorización para realizar la operación iam:PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a Lambda.

Algunos servicios de Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado marymajor intenta utilizar la consola para realizar una acción en Lambda. Sin embargo, la acción requiere que el

servicio cuenta con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

## Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de Lambda

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para obtener información acerca de si Lambda admite estas características, consulte [Cómo AWS Lambda funciona con IAM](#).
- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuenta de AWS de su propiedad, consulte [Proporcionar acceso a un usuario de IAM a otra Cuentas de AWS de la que es propietario](#) en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a tus recursos a Cuentas de AWS de terceros, consulte [Proporcionar acceso a Cuentas de AWS que son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte [Cross account resource access in IAM](#) en la Guía del usuario de IAM.

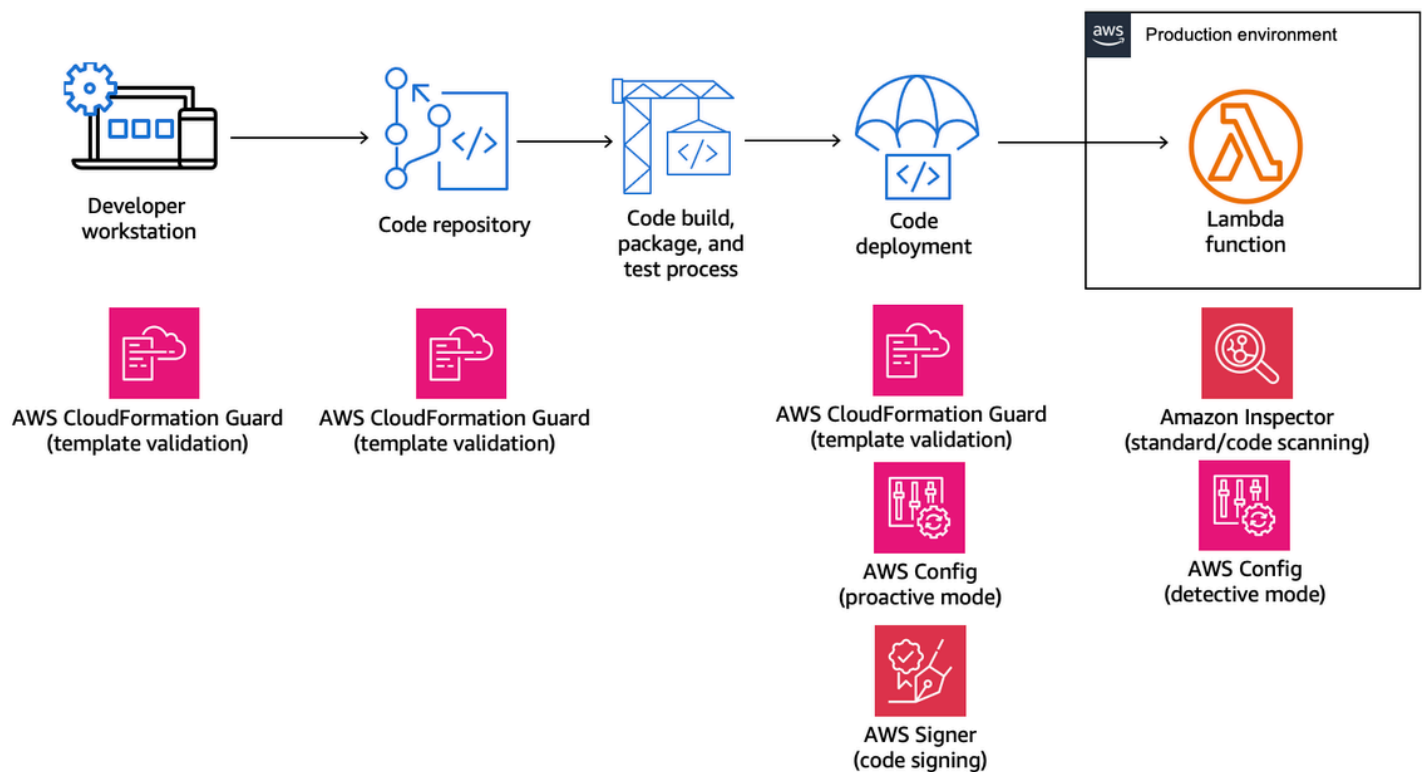
# Cree una estrategia de gobernanza para las funciones y capas de Lambda

Para crear e implementar aplicaciones nativas en la nube y sin servidor, debe permitir la agilidad y la velocidad de comercialización con la gobernanza y las barreras de protección adecuadas. Establece prioridades a nivel empresarial, tal vez haciendo hincapié en la agilidad como la máxima prioridad, o bien haciendo hincapié en la aversión al riesgo mediante la gobernanza, las barreras de protección y los controles. La realidad es que no tendrá una estrategia “lo uno o lo otro”, sino una estrategia de “y” que equilibre la agilidad y las barreras de protección en su ciclo de vida de desarrollo de software. Independientemente del lugar en el que se encuentren estos requisitos en el ciclo de vida de su empresa, es probable que las capacidades de gobernanza se conviertan en un requisito de implementación en sus procesos y cadenas de herramientas.

Estos son algunos ejemplos de controles de gobernanza que una organización podría implementar para Lambda:

- Las funciones de Lambda no deben ser accesibles públicamente.
- Las funciones de Lambda se deben adjuntar a una VPC.
- Las funciones de Lambda no deberían utilizar tiempos de ejecución obsoletos.
- Las funciones de Lambda se deben etiquetar con un conjunto de etiquetas obligatorias.
- No se debe poder acceder a las capas Lambda fuera de la organización.
- Las funciones de Lambda con un grupo de seguridad adjunto deben tener etiquetas coincidentes entre la función y el grupo de seguridad.
- Las funciones de Lambda con una capa adjunta deben utilizar una versión aprobada.
- Las variables de entorno de Lambda deben estar cifradas en reposo con una clave administrada por el cliente.

El siguiente diagrama es un ejemplo de una estrategia de gobernanza exhaustiva que implementa controles y políticas a lo largo del proceso de desarrollo e implementación del software:



En los temas siguientes se explica cómo implementar controles para desarrollar e implementar funciones de Lambda en su organización, tanto para startup como para la empresa. Es posible que su organización ya tenga las herramientas para hacerlo. En los temas siguientes se adopta un enfoque modular hacia estos controles, de modo que pueda seleccionar los componentes que realmente necesita.

## Temas

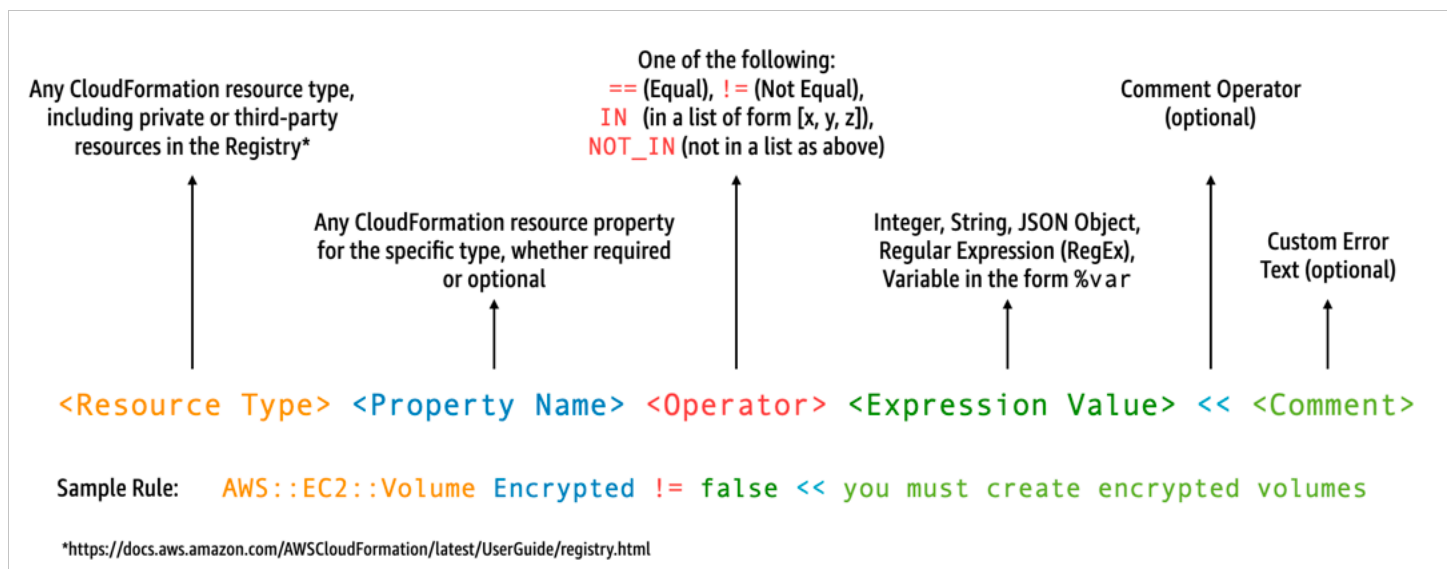
- [Controles proactivos para Lambda con AWS CloudFormation Guard](#)
- [Implemente controles preventivos para Lambda con AWS Config](#)
- [Detecte las implementaciones y configuraciones de Lambda que no cumplan con AWS Config](#)
- [Firma de código de Lambda con AWS Signer](#)
- [Automatice las evaluaciones de seguridad para Lambda con Amazon Inspector](#)
- [Implementación de la observabilidad para la seguridad y la conformidad de Lambda](#)

## Controles proactivos para Lambda con AWS CloudFormation Guard

[AWS CloudFormation Guard](#) es una herramienta de evaluación de políticas como código, de uso general y de código abierto. Se puede utilizar para la gobernanza y el cumplimiento preventivos mediante la validación del cumplimiento con las políticas de las plantillas de infraestructura como código (IaC) y las composiciones de los servicios. Estas reglas se pueden personalizar en función de los requisitos de su equipo o de la organización. En el caso de las funciones Lambda, las reglas de Guard se pueden utilizar para controlar la creación de recursos y las actualizaciones de configuración mediante la definición de los ajustes de propiedades necesarios al crear o actualizar una función de Lambda.

Los administradores de conformidad definen la lista de controles y políticas de gobernanza que se requieren para implementar y actualizar las funciones de Lambda. Los administradores de la plataforma implementan los controles en los canales de CI/CD, como webhooks de validación previa a la confirmación con repositorios de código, y proporcionan a los desarrolladores herramientas de línea de comandos para validar las plantillas y el código en las estaciones de trabajo locales. Los desarrolladores crean el código, validan las plantillas con herramientas de línea de comandos y, a continuación, envían el código a los repositorios, que luego se validan automáticamente mediante los canales de CI/CD antes de la implementación en un entorno de AWS.

Guard le permite [escribir sus reglas](#) e implementar sus controles con un lenguaje específico del dominio, como se muestra a continuación.



Por ejemplo, imagine que quiere asegurarse de que los desarrolladores elijan solo los tiempos de ejecución más recientes. Puede especificar dos políticas diferentes, una para identificar los [tiempos](#)



[de ejecución](#) que ya están en desuso y otra para identificar los tiempos de ejecución que caducarán pronto. Para ello, puede escribir el siguiente archivo de `etc/rules.guard`:

```
let lambda_functions = Resources.*[
  Type == "AWS::Lambda::Function"
]

rule lambda_already_deprecated_runtime when %lambda_functions !empty {
  %lambda_functions {
    Properties {
      when Runtime exists {
        Runtime !in ["dotnetcore3.1", "nodejs12.x", "python3.6", "python2.7",
"dotnet5.0", "dotnetcore2.1", "ruby2.5", "nodejs10.x", "nodejs8.10", "nodejs4.3",
"nodejs6.10", "dotnetcore1.0", "dotnetcore2.0", "nodejs4.3-edge", "nodejs"] <<Lambda
function is using a deprecated runtime.>>
      }
    }
  }
}

rule lambda_soon_to_be_deprecated_runtime when %lambda_functions !empty {
  %lambda_functions {
    Properties {
      when Runtime exists {
        Runtime !in ["nodejs16.x", "nodejs14.x", "python3.7", "java8",
"dotnet7", "go1.x", "ruby2.7", "provided"] <<Lambda function is using a runtime that
is targeted for deprecation.>>
      }
    }
  }
}
```

Ahora imagine que escribe la siguiente plantilla de CloudFormation de `iac/lambda.yaml` que define una función de Lambda:

```
Fn:
  Type: AWS::Lambda::Function
  Properties:
    Runtime: python3.7
    CodeUri: src
    Handler: fn.handler
    Role: !GetAtt FnRole.Arn
    Layers:
```

```
- arn:aws:lambda:us-east-1:111122223333:layer:LambdaInsightsExtension:35
```

Tras [instalar](#) la utilidad Guard, valide la plantilla:

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

El resultado tendrá este aspecto:

```
lambda.yaml Status = FAIL
FAILED rules
rules.guard/lambda_soon_to_be_deprecated_runtime
---
Evaluating data lambda.yaml against rules rules.guard
Number of non-compliant resources 1
Resource = Fn {
  Type      = AWS::Lambda::Function
  Rule = lambda_soon_to_be_deprecated_runtime {
    ALL {
      Check = Runtime not IN
["nodejs16.x", "nodejs14.x", "python3.7", "java8", "dotnet7", "go1.x", "ruby2.7", "provided"]
{
  ComparisonError {
    Message      = Lambda function is using a runtime that is targeted for
deprecation.
    Error        = Check was not compliant as property [/Resources/
Fn/Properties/Runtime[L:88,C:15]] was not present in [(resolved, Path=[L:0,C:0]
Value=["nodejs16.x", "nodejs14.x", "python3.7", "java8", "dotnet7", "go1.x", "ruby2.7", "provided"])]
  }
  PropertyPath  = /Resources/Fn/Properties/Runtime[L:88,C:15]
  Operator      = NOT IN
  Value         = "python3.7"
  ComparedWith =
["nodejs16.x", "nodejs14.x", "python3.7", "java8", "dotnet7", "go1.x", "ruby2.7", "provided"]]
  Code:
    86. Fn:
    87.   Type: AWS::Lambda::Function
    88.   Properties:
    89.     Runtime: python3.7
    90.     CodeUri: src
    91.     Handler: fn.handler
}
}
```

```
}  
}
```

Guard les permite a los desarrolladores comprobar desde sus estaciones de trabajo locales que necesitan actualizar la plantilla para utilizar un tiempo de ejecución permitido por la organización. Esto ocurre antes de iniciar sesión en un repositorio de código y no pasar las comprobaciones realizadas en los canales de CI/CD. Como resultado, los desarrolladores reciben esta información sobre cómo desarrollar plantillas compatibles y dedicar su tiempo a escribir código que aporte valor empresarial. Este control se puede aplicar en la estación de trabajo del desarrollador local, en un webhook de validación previo a la confirmación, o en el canal de CI/CD antes de la implementación.

## Advertencias

Si utiliza plantillas de AWS Serverless Application Model (AWS SAM) para definir funciones de Lambda, tenga en cuenta que debe actualizar la regla de Guard para buscar el tipo de recurso `AWS::Serverless::Function` de la siguiente manera.

```
let lambda_functions = Resources.*[  
  Type == "AWS::Serverless::Function"  
]
```

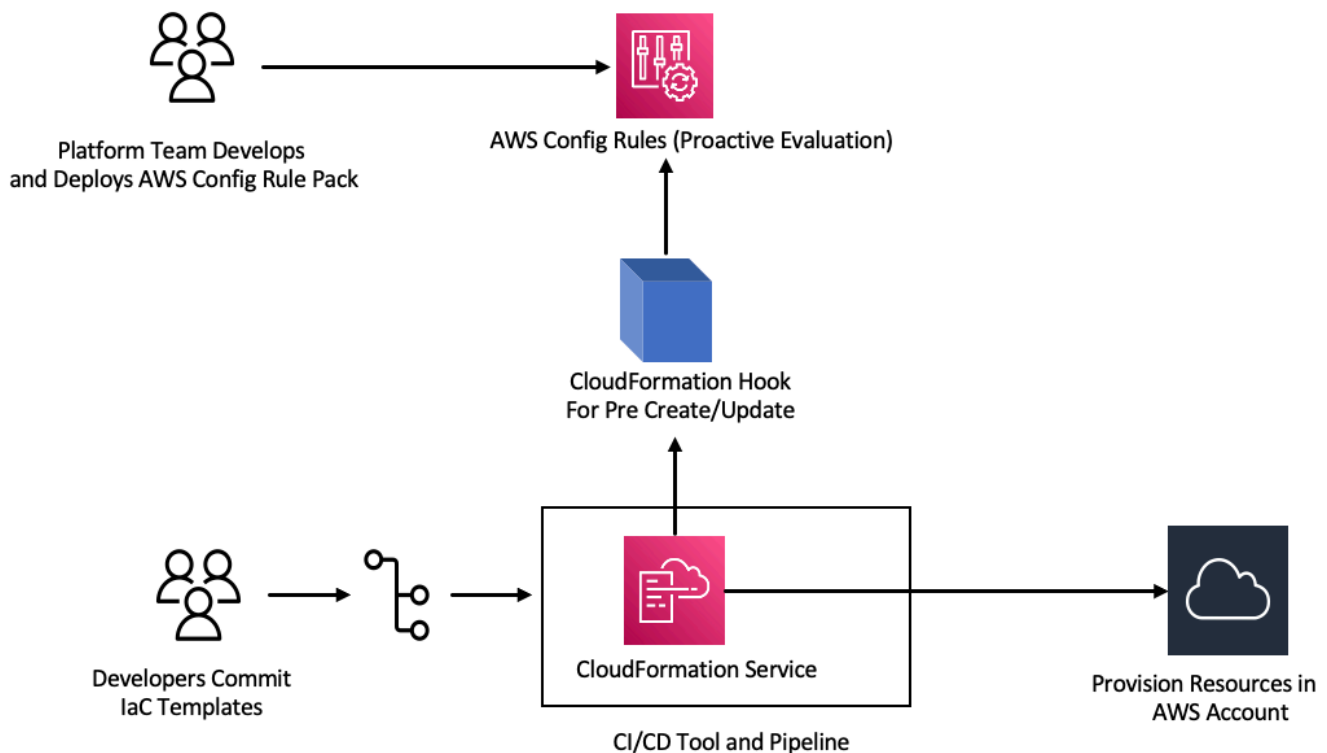
Guard también espera que las propiedades se incluyan en la definición del recurso. Mientras tanto, las plantillas de AWS SAM permiten especificar las propiedades en una sección [Global](#) independiente. Las propiedades que se definen en la sección Global no se validan con sus reglas de Guard.

Como se indica en la [documentación](#) de solución de problemas de Guard, tenga en cuenta que Guard no admite tipos intrínsecos abreviados, como `!GetAtt` o `!Sub`. Por el contrario, requiere el uso de formas expandidas, como `Fn::GetAtt` y `Fn::Sub`. (El [ejemplo anterior](#) no evalúa la propiedad `Role`, por lo que se utilizó el tipo intrínseco abreviado por simplicidad).

## Implemente controles preventivos para Lambda con AWS Config

Es esencial garantizar lo antes posible la conformidad en sus aplicaciones sin servidor durante el proceso de desarrollo. En este tema, abordamos cómo implementar controles preventivos mediante [AWS Config](#). Esto le permite implementar las comprobaciones de conformidad en una fase más temprana del proceso de desarrollo e implementar los mismos controles en sus canalizaciones de CI/CD. Esto también estandariza los controles en un repositorio de reglas administrado de forma centralizada para que pueda aplicarlos de forma coherente en todas sus cuentas de AWS.

Por ejemplo, supongamos que los administradores de conformidad definieron un requisito para garantizar que todas las funciones de Lambda incluyan el seguimiento de AWS X-Ray. Con el modo proactivo de AWS Config, puede realizar comprobaciones de conformidad en los recursos de las funciones de Lambda antes de la implementación, lo que reduce el riesgo de implementar funciones de Lambda mal configuradas y les ahorra tiempo a los desarrolladores al proporcionarles comentarios más rápidos sobre la infraestructura en forma de plantillas de código. La siguiente es una visualización del flujo de los controles preventivos con AWS Config:



Recuerde el requisito de que todas las funciones de Lambda deben tener habilitado el rastreo. En respuesta, el equipo de la plataforma identifica la necesidad de que una regla específica de AWS Config se ejecute de forma proactiva en todas las cuentas. Esta regla marca cualquier función de Lambda que carezca de una configuración de rastreo de X-Ray configurada como un recurso no conforme. El equipo desarrolla una regla, la empaqueta en un [paquete de conformidad](#) y lo implementa en todas las cuentas de AWS para garantizar que todas las cuentas de la organización apliquen estos controles de manera uniforme. Puede escribir la regla en la sintaxis AWS CloudFormation Guard 2.x.x, que adopta la siguiente forma:

```
rule name when condition { assertion }
```

A continuación, se muestra un ejemplo de regla de protección que comprueba que las funciones de Lambda tengan activado el rastreo:

```
rule lambda_tracing_check {  
  when configuration.tracingConfig exists {  
    configuration.tracingConfig.mode == "Active"  
  }  
}
```

El equipo de la plataforma toma medidas adicionales al exigir que cada implementación de AWS CloudFormation invoque un [enlace](#) de creación previa o actualización. Asumen toda la responsabilidad de desarrollar este enlace y configurar la canalización, reforzar el control centralizado de las reglas de conformidad y mantener su aplicación uniforme en todas las implementaciones. Para desarrollar, empaquetar y registrar un enlace, consulte [Desarrollo de enlaces de AWS CloudFormation](#) en la documentación de la interfaz de línea de comandos de CloudFormation (CFN-CLI). Puede usar la [CLI de CloudFormation](#) para crear el proyecto del enlace:

```
cfn init
```

Este comando le solicita información básica sobre el proyecto del enlace y crea un proyecto que incluye los siguientes archivos:

```
README.md  
<hook-name>.json  
rpdk.log  
src/handler.py  
template.yml  
hook-role.yaml
```

Como desarrollador de enlaces, debe agregar el tipo de recurso de destino deseado en el archivo de configuración de `<hook-name>.json`. En la siguiente configuración, se configura un enlace para que se ejecute antes de crear cualquier función de Lambda mediante CloudFormation. También puede agregar controladores similares para las acciones de `preUpdate` y `preDelete`.

```
"handlers": {
  "preCreate": {
    "targetNames": [
      "AWS::Lambda::Function"
    ],
    "permissions": []
  }
}
```

También debe asegurarse de que el enlace de CloudFormation tenga los permisos adecuados para llamar a las API de AWS Config. Para ello, actualice el archivo de definición de roles denominado `hook-role.yaml`. De forma predeterminada, el archivo de definición de roles posee la siguiente política de confianza, que le permite a CloudFormation asumir ese rol.

```
AssumeRolePolicyDocument:
  Version: '2012-10-17'
  Statement:
    - Effect: Allow
      Principal:
        Service:
          - hooks.cloudformation.amazonaws.com
          - resources.cloudformation.amazonaws.com
```

Para permitir que este enlace llame a las API de configuración, debe agregar los siguientes permisos a la instrucción de política. A continuación, envía el proyecto del enlace mediante el comando `cf submit`, y CloudFormation crea un rol para usted con los permisos necesarios.

```
Policies:
  - PolicyName: HookTypePolicy
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - "config:Describe*"
            - "config:Get*"
```

```

- "config:List*"
- "config:SelectResourceConfig"
Resource: "*"

```

Luego debe escribir una función de Lambda en un archivo `src/handler.py`. En este archivo, encontrará los métodos denominados `preCreate`, `preUpdate` y `preDelete`, que ya se crearon al iniciar el proyecto. Su objetivo es escribir una función común y reutilizable que llame a la API de AWS Config `StartResourceEvaluation` en modo proactivo mediante AWS SDK for Python (Boto3). Esta llamada a la API toma las propiedades del recurso como entrada y evalúa el recurso en función de la definición de la regla.

```

def validate_lambda_tracing_config(resource_type, function_properties:
MutableMapping[str, Any]) -> ProgressEvent:
    LOG.info("Fetching proactive data")
    config_client = boto3.client('config')
    resource_specs = {
        'ResourceId': 'MyFunction',
        'ResourceType': resource_type,
        'ResourceConfiguration': json.dumps(function_properties),
        'ResourceConfigurationSchemaType': 'CFN_RESOURCE_SCHEMA'
    }
    LOG.info("Resource Specifications:", resource_specs)
    eval_response = config_client.start_resource_evaluation(EvaluationMode='PROACTIVE',
ResourceDetails=resource_specs, EvaluationTimeout=60)
    ResourceEvaluationId = eval_response.ResourceEvaluationId
    compliance_response =
config_client.get_compliance_details_by_resource(ResourceEvaluationId=ResourceEvaluationId)
    LOG.info("Compliance Verification:",
compliance_response.EvaluationResults[0].ComplianceType)
    if "NON_COMPLIANT" == compliance_response.EvaluationResults[0].ComplianceType:
        return ProgressEvent(status=OperationStatus.FAILED, message="Lambda function
found with no tracing enabled : FAILED", errorCode=HandlerErrorCode.NonCompliant)
    else:
        return ProgressEvent(status=OperationStatus.SUCCESS, message="Lambda function
found with tracing enabled : PASS.")

```

Ahora puede llamar a la función común desde el controlador del enlace de creación previa. El siguiente es un ejemplo del controlador:

```

@hook.handler(HookInvocationPoint.CREATE_PRE_PROVISION)
def pre_create_handler(
    session: Optional[SessionProxy],

```

```

        request: HookHandlerRequest,
        callback_context: MutableMapping[str, Any],
        type_configuration: TypeConfigurationModel
    ) -> ProgressEvent:
        LOG.info("Starting execution of the hook")
        target_name = request.hookContext.targetName
        LOG.info("Target Name:", target_name)
        if "AWS::Lambda::Function" == target_name:
            return validate_lambda_tracing_config(target_name,
                request.hookContext.targetModel.get("resourceProperties")
            )
        else:
            raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")

```

Luego de este paso, puede registrar el enlace y configurarlo para que escuche todos los eventos de creación de las funciones de AWS Lambda.

Un desarrollador prepara la plantilla de infraestructura como código (IaC) para un microservicio sin servidor mediante Lambda. Esta preparación incluye el cumplimiento de los estándares internos, y está seguida de pruebas locales y el envío de la plantilla al repositorio. A continuación, se muestra una plantilla de IaC de ejemplo:

```

MyLambdaFunction:
  Type: 'AWS::Lambda::Function'
  Properties:
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    FunctionName: MyLambdaFunction
    Code:
      ZipFile: |
        import json

        def handler(event, context):
            return {
                'statusCode': 200,
                'body': json.dumps('Hello World!')}
    Runtime: python3.8
    TracingConfig:
      Mode: PassThrough
    MemorySize: 256
    Timeout: 10

```



Como parte del proceso de CI/CD, cuando se implementa la plantilla de CloudFormation, el servicio de CloudFormation invoca el enlace de creación previa o actualización justo antes de aprovisionar el tipo de recurso de `AWS::Lambda::Function`. El enlace utiliza reglas de AWS Config que se ejecutan en modo proactivo para comprobar si la configuración de la función de Lambda incluye la configuración de rastreo obligatoria. La respuesta del enlace determina el siguiente paso. Si es compatible, el enlace indica que se ha realizado correctamente y CloudFormation procede a aprovisionar los recursos. De lo contrario, se produce un error en la implementación de la pila de CloudFormation, la canalización se detiene inmediatamente y el sistema registra los detalles para su posterior revisión. Las notificaciones de conformidad se envían a las partes interesadas pertinentes.

Puede encontrar la información sobre el éxito o el error del enlace en la consola de CloudFormation:

Stack info	Events	Resources	Outputs	Parameters	Template	Change sets
<b>Events (19)</b>						
<input type="text" value="Search events"/>						
Timestamp	Logical ID	Status	Status reason	Hook invocations		
2023-08-29 23:50:23 UTC-0500	HookTestStack	ROLLBACK_COMPLETE	-	-		
2023-08-29 23:50:22 UTC-0500	LambdaExecutionRole	DELETE_COMPLETE	-	-		
2023-08-29 23:50:21 UTC-0500	MyApi	DELETE_COMPLETE	-	-		
2023-08-29 23:50:20 UTC-0500	LambdaExecutionRole	DELETE_IN_PROGRESS	-	-		
2023-08-29 23:50:20 UTC-0500	MyLambdaFunction	DELETE_COMPLETE	-	-		
2023-08-29 23:50:20 UTC-0500	MyApi	DELETE_IN_PROGRESS	-	-		
2023-08-29 23:50:18 UTC-0500	HookTestStack	ROLLBACK_IN_PROGRESS	The following resource(s) failed to create: [MyLambdaFunction]. Rollback requested by user.	-		
2023-08-29 23:50:17 UTC-0500	MyLambdaFunction	CREATE_FAILED	The following hook(s) failed: [AWS::Samples::LambdaTracingCheck::Hook]	-		
2023-08-29 23:50:17 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWS::Samples::LambdaTracingCheck::Hook		
2023-08-29 23:50:16 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWS::Samples::LambdaTracingCheck::Hook		
2023-08-29 23:50:15 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	-		
2023-08-29 23:50:14 UTC-0500	LambdaExecutionRole	CREATE_COMPLETE	-	-		
2023-08-29 23:49:59 UTC-0500	MyApi	CREATE_COMPLETE	-	-		
2023-08-29 23:49:59 UTC-0500	MyApi	CREATE_IN_PROGRESS	Resource creation Initiated	-		
2023-08-29 23:49:58 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	Resource creation Initiated	-		
2023-08-29 23:49:58 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	-	-		
2023-08-29 23:49:58 UTC-0500	MyApi	CREATE_IN_PROGRESS	-	-		
2023-08-29 23:49:55 UTC-0500	HookTestStack	CREATE_IN_PROGRESS	User Initiated	-		
2023-08-29 23:49:50 UTC-0500	HookTestStack	REVIEW_IN_PROGRESS	User Initiated	-		

Si tiene los registros habilitados para su enlace de CloudFormation, puede capturar el resultado de la evaluación del enlace. Este es un ejemplo de registro de un enlace con un estado de error, que indica que la función de Lambda no tiene activado X-Ray:

```

2023-08-29T23:50:17.574-05:00 ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'...

ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'>, message='Lambda
function found with no tracing enabled : FAILED', result=None, callbackContext=None, callbackDelaySeconds=0, resourceModel=None,
resourceModels=None, nextToken=None)
    
```

[Copy](#)

No newer events at this moment. *Auto retry paused.* [Resume](#)

Si el desarrollador decide cambiar la IaC para actualizar el valor `TracingConfig Mode` a `Active` y volver a implementarlo, el enlace se ejecuta correctamente y la pila continúa con la creación del recurso de Lambda.

Timestamp	Logical ID	Status	Status reason	Hook invocations
2023-08-29 23:56:52 UTC-0500	LambdaApiGatewayInvoke	CREATE_IN_PROGRESS	-	-
2023-08-29 23:56:52 UTC-0500	MyLambdaFunction	CREATE_COMPLETE	-	-
2023-08-29 23:56:44 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	Resource creation Initiated	-
2023-08-29 23:56:44 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	Hook invocations complete. Resource creation initiated	-
2023-08-29 23:56:43 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	<a href="#">AWS::Samples::LambdaTracingCheck::Hook</a>
2023-08-29 23:56:41 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	
2023-08-29 23:56:41 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	
2023-08-29 23:56:40 UTC-0500	LambdaExecutionRole	CREATE_COMPLETE	-	
2023-08-29 23:56:25 UTC-0500	MyApi	CREATE_COMPLETE	-	
2023-08-29 23:56:25 UTC-0500	MyApi	CREATE_IN_PROGRESS	Resource creation Initiated	
2023-08-29 23:56:24 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	Resource creation Initiated	
2023-08-29 23:56:23 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	-	-

**Hook invocation details**

Hook name  
[AWS::Samples::LambdaTracingCheck::Hook](#)

Hook status  
**HOOK\_COMPLETE\_SUCCEEDED**

Hook failure mode  
Fail

Hook invocation point  
PRE\_PROVISION

Hook status reason  
Hook succeeded with message: Lambda function found with tracing enabled : PASS

De esta forma, puede implementar controles preventivos con AWS Config en modo proactivo al desarrollar e implementar recursos sin servidor en sus cuentas de AWS. Al integrar las reglas de AWS Config en la canalización de CI/CD, puede identificar y, opcionalmente, bloquear las

implementaciones de recursos no conformes, como las funciones de Lambda que carecen de una configuración de rastreo activo. Esto garantiza que en sus entornos de AWS solo se implementen los recursos que cumplen con las políticas de gobernanza más recientes.

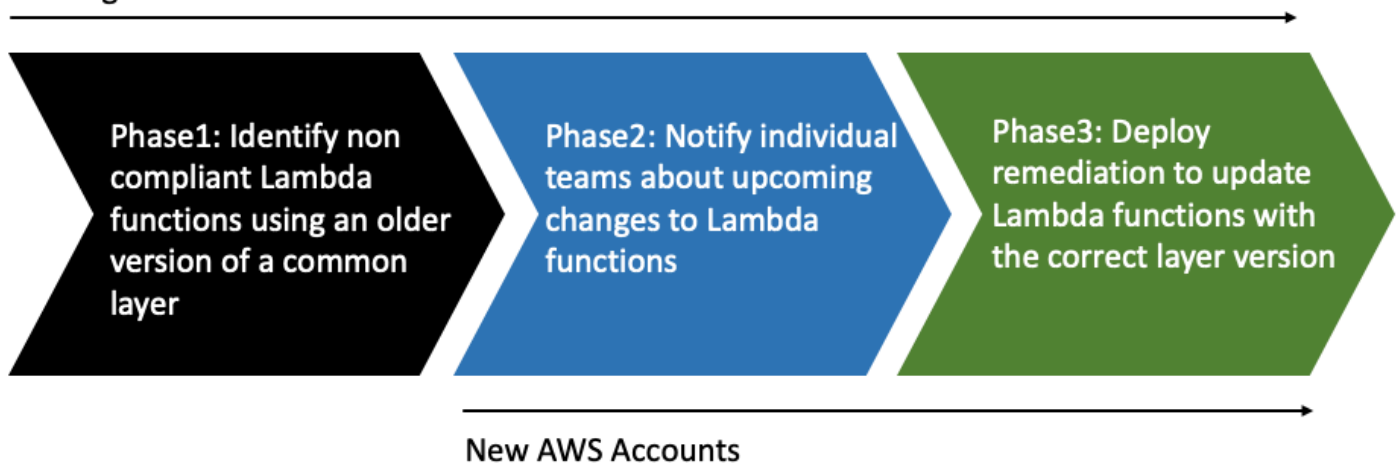
## Detecte las implementaciones y configuraciones de Lambda que no cumplan con AWS Config

Además de realizar una [evaluación proactiva](#), AWS Config también puede detectar de forma reactiva las implementaciones y configuraciones de recursos que no cumplen con sus políticas de gobernanza. Esto es importante porque las políticas de gobernanza evolucionan a medida que la organización aprende e implementa nuevas prácticas recomendadas.

Contemple un escenario en el que establezca una política completamente nueva al implementar o actualizar las funciones de Lambda: todas las funciones de Lambda deben utilizar siempre una versión de capa de Lambda específica y aprobada. Puede configurar AWS Config para supervisar las funciones nuevas o actualizadas para las configuraciones de capa. Si AWS Config detecta una función que no utiliza una versión de capa aprobada, la marca como un recurso no conforme. Si lo desea, puede configurar AWS Config para corregir automáticamente el recurso a través de la especificación de una acción de corrección mediante un documento de automatización de AWS Systems Manager. Por ejemplo, puede escribir un documento de automatización en Python mediante AWS SDK for Python (Boto3), que actualiza la función no conforme para que apunte a la versión de capa aprobada. Por lo tanto, AWS Config sirve como control de detección y de corrección, ya que automatiza la administración de la conformidad.

A continuación, desglosaremos este proceso en tres importantes fases de implementación:

### Existing AWS Accounts



### Fase 1: identificación de los recursos de acceso

Comience por activar y configurar AWS Config en todas sus cuentas para que registre las funciones de Lambda de AWS. Esto le permite a AWS Config observar cuándo se crean o actualizan las

funciones de Lambda. Luego puede configurar las [reglas de políticas personalizadas](#) para comprobar si existen infracciones específicas a las políticas, que utilizan la sintaxis AWS CloudFormation Guard. Las reglas de protección adoptan la siguiente forma general:

```
rule name when condition { assertion }
```

A continuación, se muestra un ejemplo de regla que comprueba que una capa no esté configurada en una versión de capa anterior:

```
rule desiredlayer when configuration.layers !empty {  
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn  
}
```

Para entender la sintaxis y la estructura de la regla:

- Nombre de la regla: el nombre de la regla en el ejemplo proporcionado es `desiredlayer`.
- Condición: esta cláusula especifica la condición en la que se debe comprobar la regla. En el ejemplo proporcionado, la condición es `configuration.layers !empty`. Esto significa que el recurso debe evaluarse solo cuando la propiedad `layers` de la configuración no esté vacía.
- Aserción: luego de la cláusula `when`, una aserción determina lo que comprueba la regla. La aserción `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` comprueba si alguno de los ARN de la capa de Lambda no coincide con el valor `OldLayerArn`. Si no coinciden, la aserción es verdadera y la regla se aprueba; de lo contrario, falla.

`CONFIG_RULE_PARAMETERS` es un conjunto especial de parámetros que se configura con la regla AWS Config. En este caso, `OldLayerArn` es un parámetro dentro de `CONFIG_RULE_PARAMETERS`. Esto les permite a los usuarios proporcionar un valor de ARN específico que consideren antiguo u obsoleto y, a continuación, la regla comprueba si alguna función de Lambda utiliza este ARN antiguo.

## Fase 2: visualización y diseño

AWS Config recopila datos de configuración y los almacena en buckets de Amazon Simple Storage Service (Amazon S3). Puede utilizar [Amazon Athena](#) para consultar estos datos directamente desde sus buckets de S3. Con Athena, puede agregar estos datos a nivel organizacional y generar una visión holística de las configuraciones de sus recursos en todas sus cuentas. Para configurar la agregación de los datos de configuración de recursos, consulte [Visualización de datos de AWS](#)

[Config con Athena y Amazon QuickSight](#) en el blog de Operaciones y administración de la nube de AWS.

El siguiente es un ejemplo de consulta de Athena para identificar todas las funciones de Lambda mediante un ARN de capa determinado:

```
WITH unnested AS (
  SELECT
    item.awsaccountid AS account_id,
    item.awsregion AS region,
    item.configuration AS lambda_configuration,
    item.resourceid AS resourceid,
    item.resourcename AS resourcename,
    item.configuration AS configuration,
    json_parse(item.configuration) AS lambda_json
  FROM
    default.aws_config_configuration_snapshot,
    UNNEST(configurationitems) as t(item)
  WHERE
    "dt" = 'latest'
    AND item.resourcetype = 'AWS::Lambda::Function'
)

SELECT DISTINCT
  region as Region,
  resourcename as FunctionName,
  json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
  json_extract_scalar(lambda_json, '$.timeout') AS timeout,
  json_extract_scalar(lambda_json, '$.version') AS version
FROM
  unnested
WHERE
  lambda_configuration LIKE '%arn:aws:lambda:us-
east-1:111122223333:layer:AnyGovernanceLayer:24%'
```

Estos son los resultados de la consulta:

Query results | Query stats

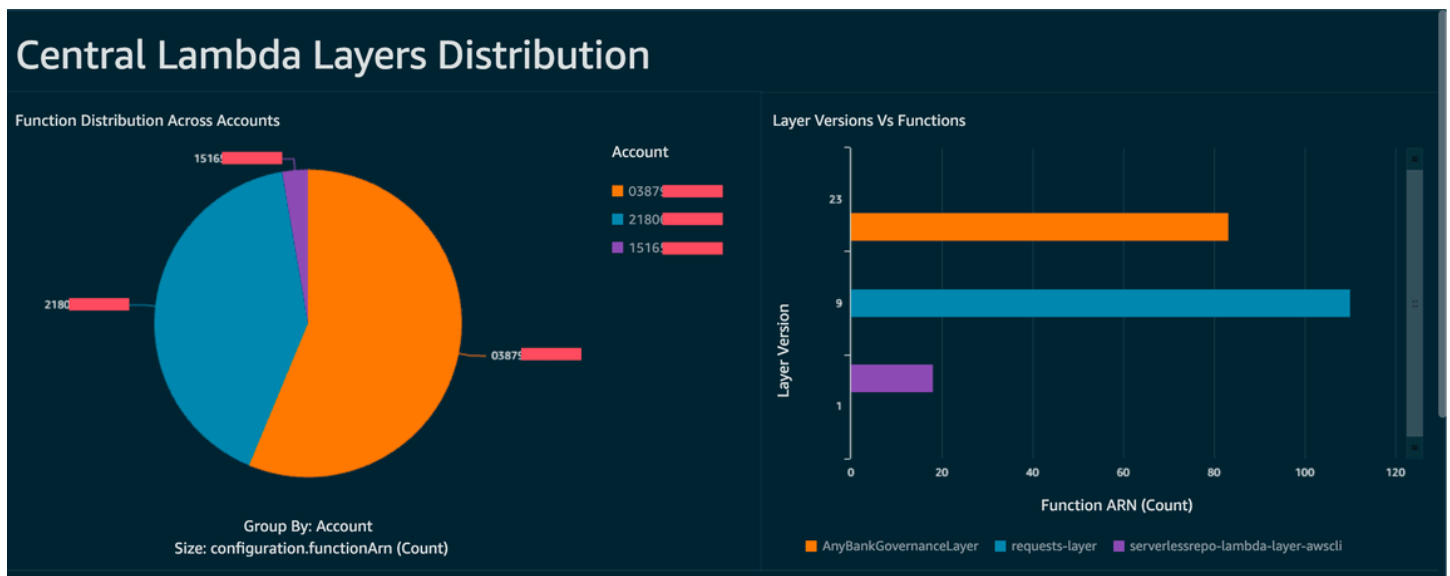
Completed Time in queue: 127 ms Run time: 1.803 sec Data scanned: 239.40 KB

Results (27) Copy Download results

Search rows

#	Region	FunctionName	memory_size	timeout	version
1	us-east-1	UpdateUIForPublishEvents	128	18	\$LATEST
2	us-east-1	SchedulerCLI-InstanceSchedulerMain	128	300	\$LATEST
3	us-east-1	my_functions_function10	128	3	\$LATEST
4	us-east-1	lex-web-ui-CognitoidentityP-CleanStackNameFunction-1TSORSH6L6YXQ	128	300	\$LATEST
5	us-east-1	GetLatestArn	128	3	\$LATEST
6	us-east-1	aws-python-http-api-project-dev-hello	1024	6	\$LATEST
7	us-east-1	cloud9-MyTest-MyTest-688JGPVYP37L	128	15	\$LATEST
8	us-east-1	my_functions_function1	128	3	\$LATEST
9	us-east-1	my_functions_function25	128	3	\$LATEST

Con los datos de AWS Config agregados en toda la organización, puede crear un panel con [Amazon QuickSight](#). Al importar los resultados de Athena a Amazon QuickSight, puede visualizar en qué medida sus funciones de Lambda cumplen la regla de versión en capas. Este panel puede destacar los recursos conformes y no conformes, lo que lo ayuda a determinar su política de cumplimiento, tal como se describe en la [siguiente sección](#). La siguiente imagen es un panel de ejemplo que informa sobre la distribución de las versiones de las capas aplicadas a las funciones de la organización.



### Fase 3: implementación y aplicación

Ahora, si lo desea, puede vincular la regla de la versión de capa que creó en la [fase 1](#) con una acción de corrección mediante un documento de automatización de Systems Manager, que puede crear como un script de Python escrito con AWS SDK for Python (Boto3). El script llama a la acción de la API [UpdateFunctionConfiguration](#) para cada función de Lambda y actualiza la configuración de la

función con el ARN de la nueva capa. Como alternativa, puede hacer que el script envíe una solicitud de extracción al repositorio del código para actualizar el ARN de la capa. De esta forma, las futuras implementaciones de código también se actualizarán con el ARN de capa correcto.



## Firma de código de Lambda con AWS Signer

[AWS Signer](#) es un servicio de firma de código totalmente gestionado que le permite validar el código con una firma digital para confirmar que el código está inalterado y que proviene de un publicador de confianza. AWS Signer se puede utilizar junto a AWS Lambda con el fin de comprobar que las funciones y las capas permanecen inalteradas antes de implementarlas en sus entornos de AWS. Esto protege a su organización de actores malintencionados que podrían haber obtenido credenciales para crear funciones nuevas o actualizar las existentes.

A fin de configurar la firma de código para las funciones de Lambda, comience por crear un bucket de S3 con el control de versiones activado. Después, cree un perfil de firma con AWS Signer, especifique Lambda como plataforma y, a continuación, especifique un período de días en el que el perfil de firma es válido. Ejemplo:

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

A continuación, utilice el perfil de firma y cree una configuración de firma con Lambda. Debe especificar qué hacer cuando la configuración de firma detecte un artefacto que no coincide con la firma digital que esperaba: avisar (pero permitir la implementación) o imponer (y bloquear la implementación). El siguiente ejemplo está configurado para imponer y bloquear las implementaciones.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

Ahora configuró AWS Signer con Lambda para bloquear las implementaciones que no son de confianza. Imagine que terminó de escribir una solicitud de característica y ahora está listo para

implementar la función. El primer paso es comprimir el código con las dependencias adecuadas y, a continuación, firmar el artefacto con el perfil de firma que creó. Para ello, puede cargar el artefacto zip en el bucket de S3 y, luego, iniciar un trabajo de firma.

```
aws signer start-signing-job \
--source 's3={bucketName=your-versioned-bucket,key=your-prefix/your-zip-artifact.zip,version=QyaJ3c4qa50LXV.9VaZgXHlsGbvCXpT}' \
--destination 's3={bucketName=your-versioned-bucket,prefix=your-prefix/}' \
--profile-name your-signer-id
```

El resultado es el siguiente: `jobId` es el objeto que se crea en el bucket y prefijo de destino y `jobOwner` es el identificador de 12 dígitos de la Cuenta de AWS en la que se ejecutó el trabajo.

```
{
  "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
  "jobOwner": "111122223333"
}
```

Y ahora puede implementar su función mediante el objeto S3 firmado y la configuración de firma de código que creó.

```
Fn:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: s3://your-versioned-bucket/your-prefix/87a3522b-5c0b-4d7d-
b4e0-4255a8e05388.zip
    Handler: fn.handler
    Role: !GetAtt FnRole.Arn
    CodeSigningConfigArn: !Ref pSigningConfigArn
```

También puede probar la implementación de una función con el artefacto zip fuente sin firmar original. La implementación debería fallar con el siguiente mensaje:

```
Lambda cannot deploy the function. The function or layer might be signed using a
signature that the client is not configured to accept. Check the provided signature
for unsigned.
```

Si está creando e implementando sus funciones mediante AWS Serverless Application Model (AWS SAM), el comando `package` se encarga de cargar el artefacto zip en S3 y también inicia el trabajo de firma y obtiene el artefacto firmado. Puede hacerlo con el comando y los parámetros que siguen:

```
sam package -t your-template.yaml \  
--output-template-file your-output.yaml \  
--s3-bucket your-versioned-bucket \  
--s3-prefix your-prefix \  
--signing-profiles your-signer-id
```

AWS Signer lo ayuda a comprobar que los artefactos zip que se implementan en sus cuentas son fiables y se pueden implementar de manera segura. Puede incluir el proceso anterior en sus canales de CI/CD y exigir que todas las funciones tengan una configuración de firma de código adjunta mediante las técnicas descritas en los temas anteriores. Si usa la firma de código en las implementaciones de funciones de Lambda, evita que actores malintencionados, que podrían haber obtenido credenciales para crear o actualizar funciones, inyecten código malicioso en sus funciones.

# Automatice las evaluaciones de seguridad para Lambda con Amazon Inspector

[Amazon Inspector](#) es un servicio de administración de vulnerabilidades que analiza de forma continua las cargas de trabajo en busca de vulnerabilidades de software y exposiciones de red no deseadas. Amazon Inspector crea un resultado en el que se describe la vulnerabilidad, se identifica el recurso afectado, se califica la gravedad de la vulnerabilidad y se proporcionan indicaciones para su corrección.

El soporte de Amazon Inspector proporciona evaluaciones de vulnerabilidades de seguridad continuas y automatizadas para las capas y funciones de Lambda. Amazon Inspector ofrece dos tipos de análisis para Lambda:

- Análisis estándar de Lambda (predeterminado): examina las dependencias de aplicaciones en una función de Lambda y sus capas en busca de [vulnerabilidades de paquetes](#).
- Análisis de código de Lambda: analiza el código personalizado de la aplicación en las funciones y sus capas en busca de [vulnerabilidades de código](#). Puede activar solo el análisis estándar de Lambda o activar este junto al análisis de código de Lambda.

Para activar Amazon Inspector, diríjase a la [consola de Amazon Inspector](#), expanda la sección Configuración y seleccione Administración de cuentas. En la pestaña Cuentas, seleccione Activar y, a continuación, seleccione una de las opciones de análisis.

Puede activar Amazon Inspector para varias cuentas y otorgar permisos para gestionar Amazon Inspector para la organización en cuentas específicas mientras configura Amazon Inspector. Mientras se habilita, debe otorgar permisos a Amazon Inspector creando el rol: `AWSServiceRoleForAmazonInspector2`. La consola de Amazon Inspector le permite crear este rol con una opción de un solo clic.

Para el análisis estándar de Lambda, Amazon Inspector inicia análisis de funciones de Lambda en busca de vulnerabilidades en las siguientes situaciones:

- En cuanto Amazon Inspector detecta una función de Lambda.
- Al implementar una nueva función de Lambda.
- cuando implementa una actualización en el código de la aplicación o las dependencias de una función de Lambda o sus capas,

- siempre que Amazon Inspector añada un nuevo elemento de vulnerabilidades y riesgos comunes (CVE) a su base de datos y ese elemento de CVE es relevante para la función.

Para el análisis de código de Lambda, Amazon Inspector evalúa el código de la aplicación de la función de Lambda mediante razonamiento automatizado y machine learning de conformidad con los estándares generales de seguridad. Si Amazon Inspector detecta una vulnerabilidad en el código de la aplicación de la función de Lambda, Amazon Inspector genera un resultado detallado del Vulnerabilidad de código. Para consultar una lista de posibles detecciones, vaya a la [Biblioteca de detectores de Amazon CodeGuru](#).

Para ver los resultados, vaya a la [Consola de Amazon Inspector](#). En el menú Resultados, seleccione Por función de Lambda para ver los resultados del análisis de seguridad que se realizaron en las funciones de Lambda.

Para excluir una función de Lambda del análisis estándar, etiquete la función con el siguiente par clave-valor:

- Key:InspectorExclusion
- Value:LambdaStandardScanning

Para excluir una función de Lambda de los análisis de código, etiquete la función con el siguiente par de clave y valor:

- Key:InspectorCodeExclusion
- Value:LambdaCodeScanning

Por ejemplo, como se muestra en la siguiente imagen, Amazon Inspector detecta automáticamente las vulnerabilidades y clasifica los resultados del tipo Code Vulnerability, lo que indica que la vulnerabilidad está en el código de la función y no en una de las bibliotecas dependientes del código. Puede comprobar estos detalles para una función específica o para varias funciones a la vez.

### Findings (2) 🔄

Choose a row to view the finding details. All findings are related to this instance.

Active ▼

Resource ID *EQUALS* `arn:aws:lambda:us-east-1:.....function:code_scanning_python:$LATEST` ✕

< 1 > ⚙️

	Severity <span>▼</span>	Title	Type <span>▼</span>	Age <span>▼</span>	Status
<input type="radio"/>	■ High	<a href="#">CWE-200 - Insecure Socket Bind</a>	Code Vulnerability	10 minutes	Active
<input type="radio"/>	■ High	<a href="#">Overriding environment variables that are res</a>	Code Vulnerability	10 minutes	Active

Puede profundizar en cada uno de estos resultados y aprender cómo solucionar el problema.

## Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior.



Finding ID: [arn:aws:inspector2:us-east-1: \[REDACTED\]:finding/\[REDACTED\]](#)

Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior or failure of the Lambda function.

### Finding overview

AWS account ID	[REDACTED]
Severity	High
Type	Code Vulnerability
Detector name <a href="#">↗</a>	<a href="#">Override of reserved variable names in a Lambda function</a>
Relevant CWE <a href="#">↗</a>	--
Rule ID <a href="#">↗</a>	<a href="#">Rule-434311</a>
Detector tags	#availability, #aws-python-sdk, #aws-lambda, #data-integrity, #maintainability, #security, #security-context, #python
Fix available	Yes
Created at	March 29, 2023 10:08 AM (UTC-04:00)

### Vulnerability details

File path `lambda_function.py`

### Vulnerability location

```
3 import socket
4
5 def lambda_handler(event, context):
6
7     # print("Scenario 1");
8     os.environ['_HANDLER'] = 'hello'
9     # print("Scenario 1 ends")
10
11     # print("Scenario 2");
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.bind(('',0))
```

### Suggested remediation

Your code attempts to override an environment variable that is reserved by the Lambda runtime environment. This can lead to unexpected behavior and might break the execution of your Lambda function.

Mientras trabaja con las funciones de Lambda, asegúrese de cumplir con las convenciones de nomenclatura de las funciones de Lambda. Para obtener más información, consulte [Utilice variables de entorno Lambda para configurar valores en el código](#).

El usuario se hace responsable de las sugerencias de corrección que acepta. Revise las sugerencias de corrección antes de aceptarlas. Es posible que deba modificar dichas sugerencias para garantizar que el código lleve a cabo las acciones previstas.



# Implementación de la observabilidad para la seguridad y la conformidad de Lambda

AWS Config es una herramienta útil para encontrar y corregir recursos sin servidor no conformes de AWS. Todos los cambios que realice en sus recursos sin servidor se registran en AWS Config. Además, AWS Config le permite almacenar los datos de las instantáneas de configuración en S3. Puede utilizar Amazon Athena y Amazon QuickSight para crear paneles y ver datos de AWS Config. En [Detecte las implementaciones y configuraciones de Lambda que no cumplan con AWS Config](#), analizamos cómo es posible visualizar una configuración determinada como capas Lambda. En este tema se amplían estos conceptos.

## Visibilidad de las configuraciones de Lambda

Puede usar consultas para obtener configuraciones importantes como el ID de Cuenta de AWS, la región, la configuración de rastreo de AWS X-Ray, la configuración de VPC, el tamaño de la memoria, el tiempo de ejecución y las etiquetas. Este es un ejemplo de consulta que puede utilizar para obtener esta información de Athena:

```
WITH unnested AS (
  SELECT
    item.awsaccountid AS account_id,
    item.awsregion AS region,
    item.configuration AS lambda_configuration,
    item.resourceid AS resourceid,
    item.resourcename AS resourcename,
    item.configuration AS configuration,
    json_parse(item.configuration) AS lambda_json
  FROM
    default.aws_config_configuration_snapshot,
    UNNEST(configurationitems) as t(item)
  WHERE
    "dt" = 'latest'
    AND item.resourcetype = 'AWS::Lambda::Function'
)

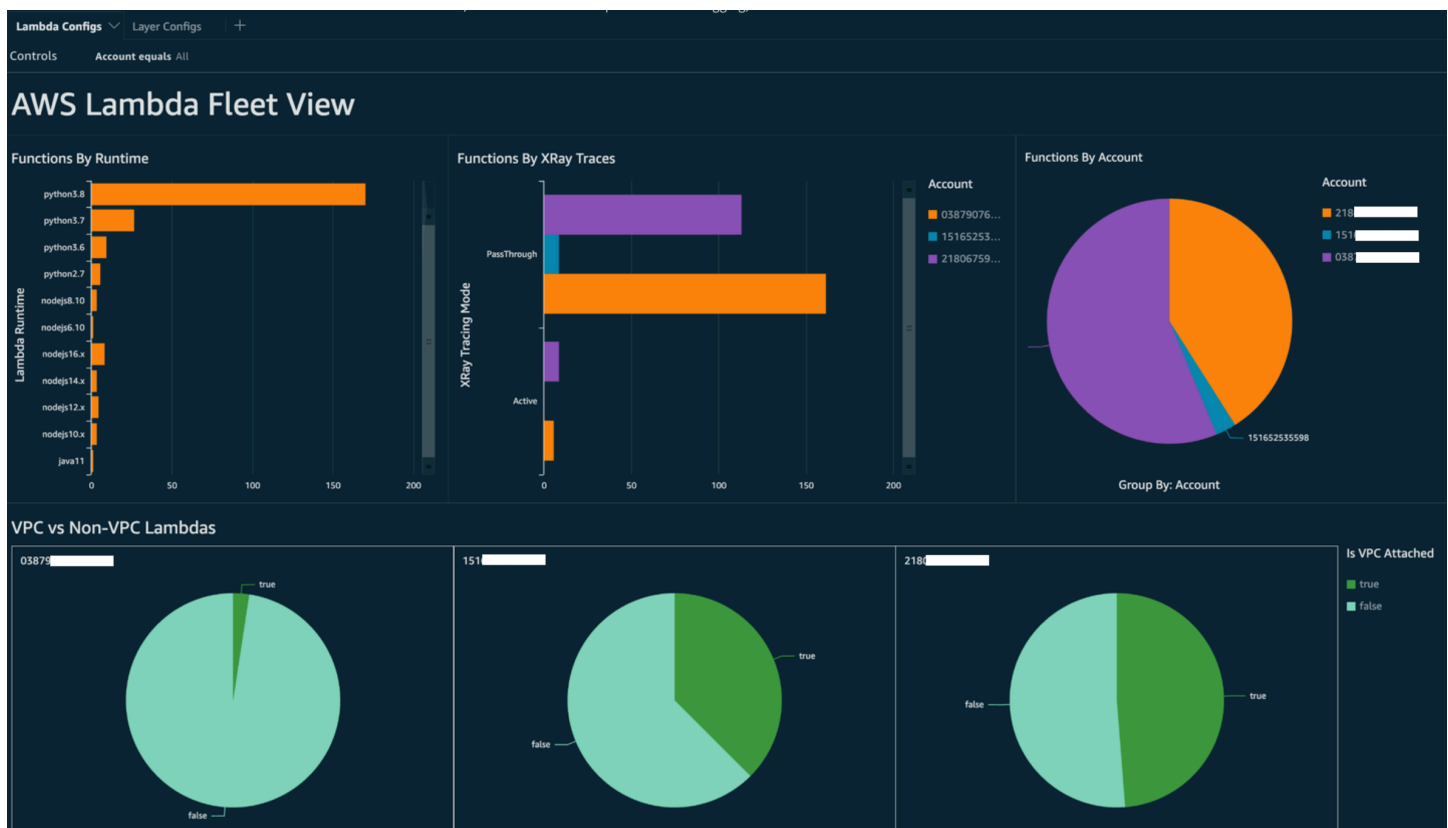
SELECT DISTINCT
  account_id,
  tags,
  region as Region,
  resourcename as FunctionName,
  json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
```

```

json_extract_scalar(lambda_json, '$.timeout') AS timeout,
json_extract_scalar(lambda_json, '$.runtime') AS version
json_extract_scalar(lambda_json, '$.vpcConfig.SubnetIds') AS vpcConfig
json_extract_scalar(lambda_json, '$.tracingConfig.mode') AS tracingConfig
FROM
  unnested

```

Puede utilizar la consulta para crear un panel de Amazon QuickSight y visualizar los datos. Para agregar datos de configuración de recursos de AWS, crear tablas en Athena y crear paneles de Amazon QuickSight a partir de los datos de Athena, consulte [Visualización de datos de AWS Config con Athena y Amazon QuickSight](#) en el blog de Operaciones y administración de la nube de AWS. Cabe destacar que esta consulta también recupera la información de las etiquetas para las funciones. Esto permite obtener información más detallada sobre sus cargas de trabajo y entornos, especialmente si emplea etiquetas personalizadas.



Para obtener más información sobre las acciones que puede realizar, consulte la sección [Cómo abordar los resultados de observabilidad](#) que aparece más adelante.

## Visibilidad de la conformidad de Lambda

Con los datos generados por AWS Config, puede crear paneles de control a nivel de organización para supervisar la conformidad. Esto permite un seguimiento y monitoreo constantes de:

- Paquetes de conformidad por puntuación de conformidad
- Reglas por recursos no conformes
- Compliance status (Estado de conformidad)

**AWS Config** ×

**Dashboard**

Conformance packs

Rules

Resources

▼ Aggregators

- Conformance packs
- Rules
- Resources
- Authorizations

Advanced queries

Settings

What's new

---

[Documentation](#)

[Partners](#)

[FAQs](#)

[Pricing](#)

[AWS Config](#) > Dashboard

## Dashboard

### Conformance Packs by Compliance Score

Conformance pack	Compliance score
MyNewConformancePack	<div style="width: 37%; height: 10px; background-color: #0070c0; border: 1px solid #ccc;"></div> 37%

### Compliance status

<p><b>Rules</b></p> <p><span style="color: red;">⚠</span> 6 Noncompliant rule(s)</p> <p><span style="color: green;">✔</span> 7 Compliant rule(s)</p>	<p><b>Resources</b></p> <p><span style="color: red;">⚠</span> 100+ Noncompliant resource(s)</p> <p><span style="color: green;">✔</span> 82 Compliant resource(s)</p>
--	--

### Noncompliant rules by noncompliant resource count

Name	Compliance
lambda-function-settings-ch...	<span style="color: red;">⚠</span> 25+ Noncompliant resource(s)
lambda-dlq-check-conforma...	<span style="color: red;">⚠</span> 25+ Noncompliant resource(s)
lambda-inside-vpc-conforma...	<span style="color: red;">⚠</span> 25+ Noncompliant resource(s)
lambda-vpc-multi-az-check-...	<span style="color: red;">⚠</span> 25+ Noncompliant resource(s)
lambda-function-settings-ch...	<span style="color: red;">⚠</span> 14 Noncompliant resource(s)

[View all noncompliant rules](#)

Compruebe cada regla para identificar los recursos no conformes con esa regla. Por ejemplo, si su organización exige que todas las funciones de Lambda estén asociadas a una VPC y si ha implementado una regla de AWS Config para identificar la conformidad, puede seleccionar la regla de `lambda-inside-vpc` en la lista anterior.

**Resources in scope**

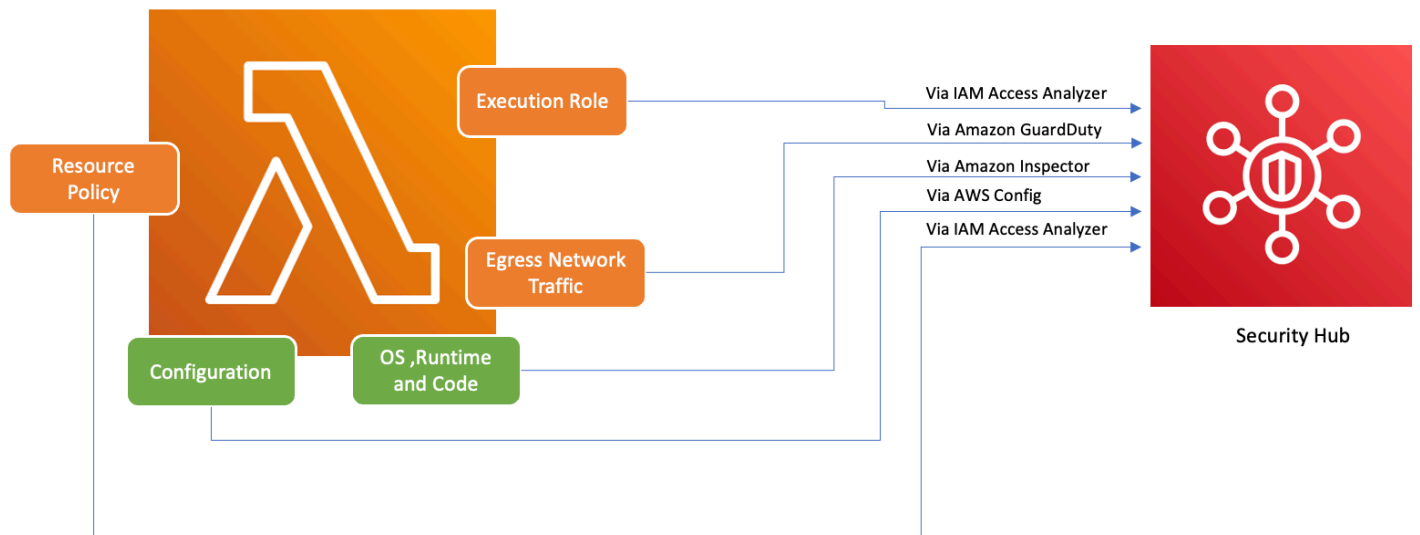
All

- All
- Compliant
- Noncompliant

	Type	Annotation	Compliance
<input type="radio"/> my_functions_function44	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function46	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function47	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function49	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function50	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function6	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function7	Lambda Function	-	✔ Compliant
<input type="radio"/> my_functions_function8	Lambda Function	-	✔ Compliant
<input type="radio"/> ConfigQueryLambda	Lambda Function	This AWS Lambda function is not in ...	⚠ Noncompliant
<input type="radio"/> DormamuLambda	Lambda Function	This AWS Lambda function is not in ...	⚠ Noncompliant

Para obtener más información sobre las acciones que puede realizar, consulte la sección [Cómo abordar los resultados de observabilidad](#) que aparece más adelante.

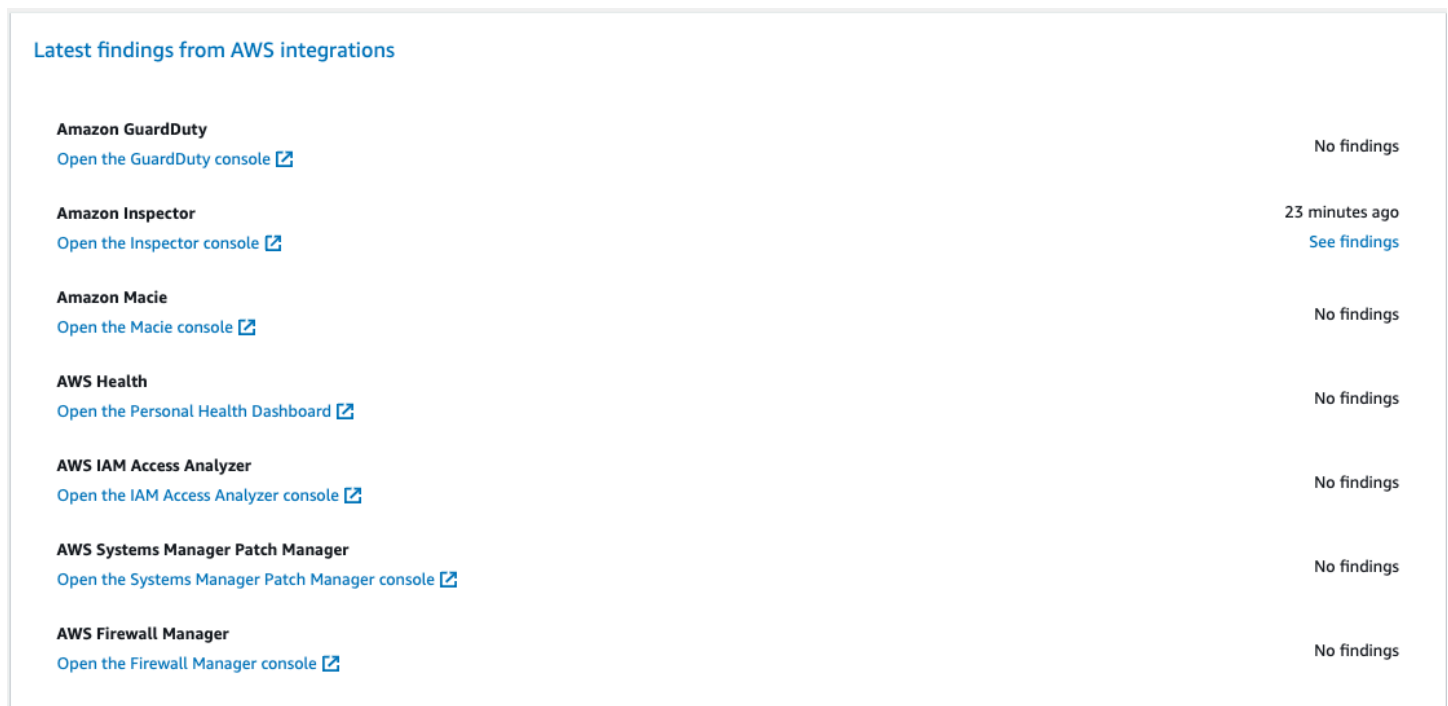
### Visibilidad de los límites de las funciones de Lambda mediante Security Hub



Para garantizar que los servicios de AWS, incluyendo Lambda, se utilicen de forma segura, AWS incorporó las Prácticas recomendadas de seguridad fundamentales v1.0.0. Este conjunto de prácticas recomendadas proporciona directrices claras para proteger los recursos y los datos en el entorno de AWS, y hace hincapié en la importancia de mantener una postura de seguridad sólida. AWS Security Hub complementa esto al ofrecer un centro unificado de seguridad y conformidad.

Reúne, organiza y prioriza los resultados de seguridad de varios servicios de AWS, como Amazon Inspector, AWS Identity and Access Management Access Analyzer y Amazon GuardDuty.

Si tiene habilitados Security Hub, Amazon Inspector, IAM Access Analyzer y GuardDuty en su organización de AWS, Security Hub agrega automáticamente los resultados de estos servicios. Por ejemplo, a continuación analizaremos Amazon Inspector. Con Security Hub, puede identificar de manera eficiente las vulnerabilidades de código y paquete en las funciones de Lambda. En la consola de Security Hub, diríjase a la sección inferior denominada Últimos resultados de las integraciones de AWS. Allí puede ver y analizar los resultados obtenidos de varios servicios integrados de AWS.



Latest findings from AWS integrations	
<b>Amazon GuardDuty</b> <a href="#">Open the GuardDuty console</a>	No findings
<b>Amazon Inspector</b> <a href="#">Open the Inspector console</a>	23 minutes ago <a href="#">See findings</a>
<b>Amazon Macie</b> <a href="#">Open the Macie console</a>	No findings
<b>AWS Health</b> <a href="#">Open the Personal Health Dashboard</a>	No findings
<b>AWS IAM Access Analyzer</b> <a href="#">Open the IAM Access Analyzer console</a>	No findings
<b>AWS Systems Manager Patch Manager</b> <a href="#">Open the Systems Manager Patch Manager console</a>	No findings
<b>AWS Firewall Manager</b> <a href="#">Open the Firewall Manager console</a>	No findings

Para ver los detalles, seleccione el enlace Ver los resultados en la segunda columna. Se mostrará una lista de resultados filtrados por producto, como Amazon Inspector. Para limitar la búsqueda a las funciones de Lambda, establezca `ResourceType` en `AwsLambdaFunction`. Se mostrarán los resultados de Amazon Inspector relacionados con las funciones de Lambda.

Security Hub > Findings

**Findings (20+)** Actions Workflow status Create insight

A finding is a security issue or a failed security check.

Q Add filter

Product name is Inspector X Resource type is AwsLambdaFunction X Workflow status is NEW X Workflow status is NOTIFIED X Record state is ACTIVE X Clear filters

< 1 ... >

<input type="checkbox"/>	Severity	Workflow status	Record State	Region	Account Id	Company	Product	Title	Resource	Compliance Status	Updated at
<input type="checkbox"/>	HIGH	NEW	ACTIVE	us-east-1	218	Amazon	Inspector	CWE-117 - Log injection	Lambda Function \$LATEST		27 minutes ago
<input type="checkbox"/>	HIGH	NEW	ACTIVE	us-east-1	218	Amazon	Inspector	CWE-117 - Log injection	Lambda Function \$LATEST		27 minutes ago
<input type="checkbox"/>	HIGH	NEW	ACTIVE	us-east-1	218	Amazon	Inspector	CWE-117 - Log injection	Lambda Function \$LATEST		27 minutes ago
<input type="checkbox"/>	HIGH	NEW	ACTIVE	us-east-1	218	Amazon	Inspector	CWE-117 - Log injection	Lambda Function \$LATEST		27 minutes ago

En el caso de GuardDuty, puede identificar patrones de tráfico de red sospechosos. Estas anomalías pueden sugerir la existencia de código potencialmente malicioso en la función de Lambda.

Con IAM Access Analyzer, puede comprobar las políticas, especialmente las que contienen declaraciones de condiciones que permiten que las funciones accedan a entidades externas. Además, IAM Access Analyzer evalúa los permisos establecidos al utilizar la operación [AddPermission](#) en la API de Lambda junto con un EventSourceToken.

## Cómo abordar los resultados de observabilidad

Dada la amplia gama de configuraciones posibles para las funciones de Lambda y sus diferentes requisitos, es posible que una solución de automatización estandarizada para la corrección no se adapte a todas las situaciones. Además, los cambios se implementan de manera diferente en cada entorno. Si encuentra alguna configuración que parece no cumplir con los requisitos, tenga en cuenta las siguientes pautas:

### 1. Estrategia de etiquetado

Recomendamos implementar una estrategia de etiquetado integral. Cada función de Lambda debe etiquetarse con información clave, tal como:

- Propietario: la persona o el equipo responsable de la función.
- Entorno: producción, staging, desarrollo o entorno aislado.

- Aplicación: el contexto más amplio al que pertenece esta función, si corresponde.

## 2. Contacto con los propietarios

En lugar de automatizar los cambios importantes (como el ajuste de la configuración de la VPC), póngase en contacto de forma proactiva con los propietarios de las funciones no conformes (identificadas con la etiqueta de propietario), quienes, de esta manera, tendrán tiempo suficiente para:

- Ajustar las configuraciones no conformes en las funciones de Lambda.
- Proporcionar una explicación y solicitar una excepción, o perfeccionar los estándares de conformidad.

## 3. Mantenimiento de una base de datos de la administración de la configuración (CMDB).

Si bien las etiquetas pueden proporcionar un contexto inmediato, el mantenimiento de una CMDB centralizada puede proporcionar información más detallada. Puede contener información más minuciosa sobre cada función de Lambda, sus dependencias y otros metadatos importantes. Una CMDB es un recurso indispensable para realizar auditorías, comprobar la conformidad e identificar a los propietarios de las funciones.

A medida que el panorama de la infraestructura sin servidor comienza a evolucionar continuamente, es esencial adoptar una postura proactiva con respecto al monitoreo. Con herramientas como AWS Config, Security Hub y Amazon Inspector, se pueden identificar rápidamente posibles anomalías o configuraciones no conformes. Sin embargo, las herramientas por sí solas no pueden garantizar el cumplimiento total ni las configuraciones óptimas. Es crucial combinar estas herramientas con prácticas recomendadas y procesos bien documentados.

- Bucle de retroalimentación: una vez que se hayan tomado las medidas correctivas, asegúrese de que haya un bucle de retroalimentación. Esto implica visitar periódicamente los recursos no conformes para verificar si se han actualizado o si siguen funcionando con los mismos problemas.
- Documentación: documente siempre las observaciones, las medidas adoptadas y las excepciones concedidas. La documentación adecuada no solo ayuda durante las auditorías, sino que también ayuda a mejorar el proceso para mejorar la conformidad y la seguridad en el futuro.
- Capacitación y concientización: asegúrese de que todas las partes interesadas, especialmente los propietarios de las funciones de Lambda, reciban capacitación periódica y conozcan las prácticas recomendadas, las políticas organizacionales y los mandatos de conformidad. Los talleres, las



sesiones de formación y los seminarios web periódicos pueden contribuir en gran medida a garantizar que todos estén de acuerdo en lo que respecta a la seguridad y la conformidad.

En conclusión, si bien las herramientas y las tecnologías ofrecen capacidades sólidas para detectar y marcar posibles problemas, el elemento humano (comprensión, comunicación, formación y documentación) sigue siendo fundamental. Se combinan para formar un conjunto poderoso y así garantizar que sus funciones de Lambda y su infraestructura más amplia sigan cumpliendo con las normas, sean seguras y estén optimizadas para las necesidades de su empresa.

## Validación de conformidad en AWS Lambda

Audidores externos evalúan la seguridad y la conformidad de AWS Lambda como parte de varios programas de conformidad de AWS. Estos incluyen SOC, PCI, FedRAMP, HIPAA y otros.

Para obtener una lista de AWS servicios en el ámbito de programas de cumplimiento específicos, consulte los [AWS servicios en ámbito por programa de cumplimiento](#). Para obtener información general, consulte [Programas de conformidad de AWS](#).

Puede descargar los informes de auditoría de terceros con AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#).

Su responsabilidad de conformidad cuando usa Lambda se determina en función de la confidencialidad de los datos, los objetivos de conformidad de su empresa, la legislación y los reglamentos aplicables. Puede implementar controles de gobierno para garantizar que las funciones de Lambda de su empresa cumplan con los requisitos de conformidad. Para obtener más información, consulte [Cree una estrategia de gobernanza para las funciones y capas de Lambda](#).

## Resiliencia en AWS Lambda

La infraestructura global de AWS se compone de regiones de AWS y zonas de disponibilidad de AWS. Las regiones proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre las zonas de disponibilidad y las regiones de AWS, consulte [Infraestructura global de AWS](#).

Además de la infraestructura global de AWS, Lambda ofrece varias características que le ayudan con sus necesidades de resiliencia y copia de seguridad de los datos.

- **Control de versiones:** Puede utilizar el control de versiones en Lambda para guardar el código de la función y la configuración a medida que realice el desarrollo. Junto con los alias, puede utilizar el control de versiones para realizar implementaciones continuas y blue/green. Para obtener más detalles, consulte [Administrar las versiones de la función de Lambda](#).

- **Escalado:** cuando la función recibe una solicitud mientras se procesa una solicitud anterior, Lambda lanza otra instancia de su función para administrar el aumento de la carga. Lambda se escala automáticamente para manejar 1000 ejecuciones simultáneas por región, una [cuota](#) que se pueden aumentar si es necesario. Para obtener más detalles, consulte [Comprender el escalado de la función de Lambda](#).
- **Alta disponibilidad:** Lambda ejecuta la función en varias zonas de disponibilidad para asegurarse de que está disponible para procesar eventos en caso de una interrupción del servicio en una sola zona. Si configura la función para conectarse a una nube privada virtual (VPC) en su cuenta, especifique subredes en varias zonas de disponibilidad para garantizar una alta disponibilidad. Para obtener más detalles, consulte [Otorgamiento a las funciones de Lambda de acceso a los recursos de una Amazon VPC](#).
- **Simultaneidad reservada:** para asegurarse de que la función siempre puede escalarse para gestionar solicitudes adicionales, puede reservar la simultaneidad para ella. Configurar la simultaneidad reservada para una función garantiza que se puede escalar, pero no superar, un determinado número de invocaciones simultáneas. De este modo, se asegura de que no pierde las solicitudes debido a otras funciones que consumen toda la simultaneidad disponible. Para obtener más detalles, consulte [Configurar la simultaneidad reservada para una función](#).
- **Reintentos:** en invocaciones asíncronas y un subconjunto de invocaciones activadas por otros servicios, Lambda reintentará automáticamente en error con retrasos entre los reintentos. Otros clientes y servicios de AWS que invocan de forma sincrónica son responsables de realizar los reintentos. Para obtener más detalles, consulte [Comprender el comportamiento de reintento en Lambda](#).
- **Cola de mensajes fallidos:** en invocaciones asíncronas, puede configurar Lambda para enviar solicitudes a una cola de mensajes fallidos si fallan todos los reintentos. Una cola de mensajes fallidos es un tema de Amazon SNS o una cola de Amazon SQS que recibe eventos para la resolución de problemas o para reprocesamiento. Para obtener más información, consulte [Colas de mensajes fallidos](#).

## Seguridad de la infraestructura en AWS Lambda

Como se trata de un servicio administrado, AWS Lambda está protegido por la seguridad de red global de AWS. Para obtener información sobre los servicios de seguridad de AWS y cómo AWS protege la infraestructura, consulte [Seguridad en la nube de AWS](#). Para diseñar su entorno de AWS con las prácticas recomendadas de seguridad de infraestructura, consulte [Protección de la infraestructura](#) en Portal de seguridad de AWS Well-Architected Framework.

Puede utilizar llamadas a la API publicadas en AWS para obtener acceso a Lambda a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad de seguridad de IAM principal. También puede utilizar [AWS Security Token Service](#) (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

## Uso de la firma de código para verificar la integridad del código con Lambda

La firma de código de AWS Lambda ayuda a garantizar que solo se ejecute código de confianza en sus funciones de Lambda. Cuando habilita la firma de código para una función, Lambda comprueba todas las implementaciones de código y comprueba que el paquete de código está firmado por una fuente de confianza.

### Note

Las funciones definidas como imágenes de contenedor no admiten la firma de código.

Para verificar la integridad del código, utilice [AWS Signer](#) para crear paquetes de código firmados digitalmente para funciones y capas. Cuando un usuario intenta implementar un paquete de código, Lambda realiza comprobaciones de validación en el paquete de código antes de aceptar la implementación. Dado que las comprobaciones de validación de firma de código se ejecutan en tiempo de implementación, no hay impacto en el rendimiento en la ejecución de funciones.

También se utiliza AWS Signer para crear perfiles de firma. Utilice un perfil de firma para crear el paquete de código firmado. Utilice AWS Identity and Access Management (IAM) para controlar quién puede firmar paquetes de código y crear perfiles de firma. Para obtener más información, consulte [Autenticación y control de acceso](#) en la Guía del desarrollador de AWS Signer.

Lambda siguen el mismo formato de paquete de código firmado que los paquetes de código de función. Cuando agrega una capa a una función que tiene habilitada la firma de código, Lambda comprueba que la capa está firmada por un perfil de firma permitido. Cuando habilita la firma de código para una función, todas las capas que se agregan a la función también deben estar firmadas por uno de los perfiles de firma permitidos.

Puede configurar la firma de código para registrar los cambios en AWS CloudTrail. Las implementaciones exitosas y bloqueadas en funciones se registran en CloudTrail con información sobre las comprobaciones de firma y validación.

No hay ningún cargo adicional por usar AWS Signer o firma de código para AWS Lambda.

## Validación de firmas

Lambda realiza las siguientes comprobaciones de validación cuando implementa un paquete de código firmado en su función:

1. **Integridad:** valida que el paquete de código no se ha modificado desde que se firmó. Lambda compara el hash del paquete con el hash de la firma.
2. **Caducidad:** valida que la firma del paquete de código no ha caducado.
3. **No coincidencia:** valida que el paquete de código está firmado con uno de los perfiles de firma permitidos para la función de Lambda. También se produce una falta de coincidencia si una firma no está presente.
4. **Revocación:** valida que la firma del paquete de código no se ha revocado.

La directiva de validación de firmas definida en la configuración de firma de código determina cuál de las siguientes acciones de Lambda se lleva a cabo si falla alguna de las comprobaciones de validación:

- **Advertencia:** Lambda permite la implementación del paquete de código, pero emite una advertencia. Lambda emite una nueva métrica de Amazon CloudWatch y también almacena la advertencia en el registro de CloudTrail.
- **Forzar:** Lambda emite una advertencia (lo mismo que para la acción Advertencia) y bloquea la implementación del paquete de código.

Puede configurar la directiva para las comprobaciones de validación de caducidad, discordancia y revocación. Tenga en cuenta que no puede configurar una directiva para la comprobación de integridad. Si falla la comprobación de integridad, Lambda bloquea la implementación.

## Configuración de la firma de código con la API de Lambda

Para administrar configuraciones de firma de código con AWS CLI o AWS SDK, utilice las siguientes operaciones de API:

- [ListCodeSigningConfigs](#)
- [CreateCodeSigningConfig](#)
- [GetCodeSigningConfig](#)
- [UpdateCodeSigningConfig](#)
- [DeleteCodeSigningConfig](#)

Para administrar la configuración de firma de código de una función, utilice las siguientes operaciones de API:

- [CreateFunction](#)
- [GetFunctionCodeSigningConfig](#)
- [PutFunctionCodeSigningConfig](#)
- [DeleteFunctionCodeSigningConfig](#)
- [ListFunctionsByCodeSigningConfig](#)

### Temas

- [Creación de configuraciones de firma de código para Lambda](#)
- [Actualización de una configuración de firma de código](#)
- [Configuración de las políticas de IAM para configuraciones de firma de código de Lambda](#)

## Creación de configuraciones de firma de código para Lambda

Para habilitar la firma de código para una función, debe crear una configuración de firma de código y adjuntarla a la función. Una configuración de firma de código define una lista de perfiles de firma

permitidos y la acción de directiva que se debe realizar si falla alguna de las comprobaciones de validación.

## Secciones

- [Requisitos previos de configuración](#)
- [Creación de configuraciones de firma de código](#)
- [Habilitar la firma de código para una función](#)

## Requisitos previos de configuración

Antes de configurar la firma de código para una función de Lambda, utilice AWS Signer para hacer lo siguiente:

- Cree uno o varios perfiles de firma.
- Utilice un perfil de firma para crear un paquete de código firmado para su función.

Para obtener más información, consulte [Creación de perfiles de firma \(Console\)](#) en la Guía del desarrollador de AWS Signer.

## Creación de configuraciones de firma de código

Una configuración de firma de código define una lista de perfiles de firma permitidos y la directiva de validación de firmas.

Para crear una configuración de firma de código (consola)

1. Abra la página [Configuraciones de firma de código](#) de la consola de Lambda.
2. Seleccione Crear configuración.
3. En Descripción, introduzca un nombre descriptivo para la configuración.
4. En Perfiles de firma, agregue hasta 20 perfiles de firma a la configuración.
  - a. Para Firmar versión de perfil ARN, elija el Nombre de recurso de Amazon (ARN) de una versión de perfil o introduzca el ARN.
  - b. Para agregar un perfil de firma adicional, elija Agregar perfiles de firma.
5. En Política de validación de firmas, seleccione Advertir o Aplicar.
6. Seleccione Crear configuración.

## Habilitar la firma de código para una función

Para habilitar la firma de código para una función, asocie una configuración de firma de código con la función.

Para asociar una configuración de firma de código con una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función para la que desea habilitar la firma de código.
3. Haga clic en la pestaña Configuration (Configuración).
4. Desplácese hacia abajo y seleccione Firma de código.
5. Elija Editar.
6. En Editar firma de código, elija una configuración de firma de código para esta función.
7. Seleccione Guardar.

## Actualización de una configuración de firma de código

Cuando actualiza una [configuración de firma de código](#), los cambios afectan a las futuras implementaciones de funciones que tienen adjunta la configuración de firma de código.

Para actualizar una configuración de firma de código (consola)

1. Abra la página [Configuraciones de firma de código](#) de la consola de Lambda.
2. Seleccione una configuración de firma de código para actualizar y, a continuación, elija Editar.
3. En Descripción, introduzca un nombre descriptivo para la configuración.
4. En Perfiles de firma, agregue hasta 20 perfiles de firma a la configuración.
  - a. Para Firmar versión de perfil ARN, elija el Nombre de recurso de Amazon (ARN) de una versión de perfil o introduzca el ARN.
  - b. Para agregar un perfil de firma adicional, elija Agregar perfiles de firma.
5. En Política de validación de firmas, seleccione Advertir o Aplicar.
6. Elija Guardar cambios.



## Configuración de las políticas de IAM para configuraciones de firma de código de Lambda

Para conceder permiso a un usuario para acceder a [las operaciones de la API de firma de código](#), adjunte una o varias instrucciones de directiva a la directiva de usuario. Para obtener más información sobre las políticas de usuario, consulte [Políticas de IAM basadas en identidad para Lambda](#).

En la siguiente declaración de directiva de ejemplo se concede permiso para crear, actualizar y recuperar configuraciones de firma de código.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateCodeSigningConfig",
        "lambda:UpdateCodeSigningConfig",
        "lambda:GetCodeSigningConfig"
      ],
      "Resource": "*"
    }
  ]
}
```

Los administradores pueden usar la clave de `CodeSigningConfigArn` condición para especificar las configuraciones de firma de código que los desarrolladores deben utilizar para crear o actualizar sus funciones.

En la siguiente declaración de política de ejemplo se concede permiso para crear una función. La declaración de política incluye una condición `lambda:CodeSigningConfigArn` para especificar la configuración de firma de código permitida. Lambda bloquea cualquier solicitud de API `CreateFunction` si su parámetro `CodeSigningConfigArn` no está o no coincide con el valor de la condición.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
"Sid": "AllowReferencingCodeSigningConfig",
"Effect": "Allow",
"Action": [
    "lambda:CreateFunction",
],
"Resource": "*",
"Condition": {
    "StringEquals": {
        "lambda:CodeSigningConfigArn":
            "arn:aws:lambda:us-west-2:123456789012:code-signing-
config:csc-0d4518bd353a0a7c6"
    }
}
]
```

# Supervisión y solución de problemas de funciones de Lambda

AWS Lambda se integra con otros servicios de AWS para ayudarlo a monitorear y solucionar problemas de sus funciones de Lambda. Lambda supervisa automáticamente las funciones de Lambda en nombre y los informes de métricas a través de Amazon CloudWatch. Para ayudarlo a monitorear su código cuando se ejecuta, Lambda realiza un seguimiento automáticamente del número de solicitudes, la duración de la invocación por solicitud y el número de solicitudes que dan lugar a un error.

Puede utilizar otros servicios de AWS para solucionar problemas de sus funciones de Lambda. En esta sección se describe cómo utilizar estos servicios de AWS para la monitorización, rastreo, eliminación de fallas y solución de problemas de sus funciones y aplicaciones de Lambda. Para obtener más información sobre el registro de funciones y los errores en cada tiempo de ejecución, consulte las secciones de cada tiempo de ejecución.

Para obtener más información sobre la supervisión de las aplicaciones de Lambda, consulte [Monitoring and observability](#) en Serverless Land.

## Secciones

- [Precios](#)
- [Supervisión de funciones en la consola de Lambda](#)
- [Ver las métricas de funciones de Lambda](#)
- [Uso de Registros de Amazon CloudWatch con AWS Lambda](#)
- [Registrar llamadas a la API de AWS Lambda mediante AWS CloudTrail](#)
- [Visualice las invocaciones de la función de Lambda mediante AWS X-Ray](#)
- [Supervise el rendimiento de las funciones con Amazon CloudWatch Lambda Insights](#)

## Precios

CloudWatch una capa gratuita permanente. Más allá del umbral de capa gratuita, CloudWatch hace cargos por métricas, paneles, alarmas, registros e información. Para más información, consulte [Precios de Amazon CloudWatch](#).

# Supervisión de funciones en la consola de Lambda

El servicio de Lambda se encarga de supervisar las funciones en su nombre y enviar métricas a Amazon CloudWatch. La consola de Lambda crea gráficos de monitorización para estas métricas y las muestra en la página Monitorización de cada función de Lambda.

La consola de Lambda proporciona una vista de panel único de métricas, registros y trazas. La consola proporciona filtros para el rango horario, la zona horaria y las opciones de actualización que se aplican a todos los paneles de forma universal. Puede correlacionar fácilmente las métricas, los registros y los registros de seguimiento, lo que reduce el tiempo medio de recuperación (MTTR) a la hora de solucionar errores en las funciones de Lambda.

## Uso de la consola de Lambda

Puede supervisar las funciones y aplicaciones de Lambda en la consola de Lambda.

Para monitorizar una función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Monitor (Monitorear).

## Tipos de gráficos de monitorización

En la siguiente sección se describen los gráficos de monitorización de la consola de Lambda.

Gráficos de monitorización de Lambda

- **Invocaciones:** el número de veces que se invocó la función.
- **Duration (Duración):** la cantidad de tiempo promedio, mínimo y máximo que el código de la función dedica a procesar un evento.
- **Error count and success rate (%) (Número de errores y tasa de éxito [%]):** número de errores y porcentaje de invocaciones que se completaron sin errores.
- **Throttles (Limitaciones):** número de veces que se produjo un error en una invocación debido a los límites de simultaneidad.
- **IteratorAge:** en los orígenes de eventos de flujo, la antigüedad del último elemento del lote cuando Lambda lo recibe e invoca la función.

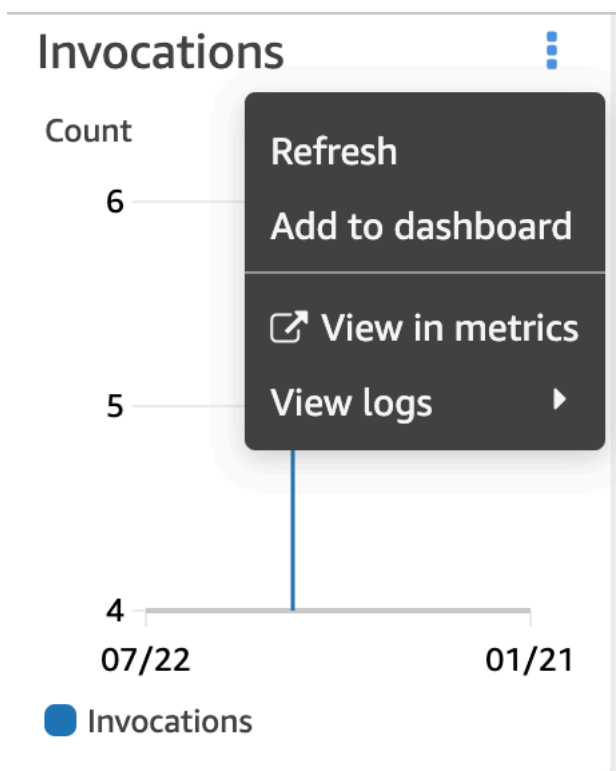
- Errores de entrega asincrónicos: número de errores que se produjeron cuando Lambda intentó escribir en una cola de mensajes fallidos o destino.
- Ejecuciones simultáneas: número de instancias de función que están procesando eventos.

## Visualización de gráficos en la consola de Lambda

En la siguiente sección se describe cómo ver gráficos de monitorización de CloudWatch en la consola de Lambda y abrir el panel de métricas de CloudWatch.

Para ver gráficos de monitorización de una función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Monitor (Monitorear).
4. Elija entre los intervalos de tiempo predefinidos o elija un intervalo de tiempo personalizado.
5. Para ver la definición de un gráfico en CloudWatch, seleccione los tres puntos verticales (Widget actions [Acciones de widget]) y luego seleccione View in metrics (Ver en métricas) para abrir el panel Metrics (Métricas) en la consola de CloudWatch.



## Visualización de consultas en la consola de CloudWatch Logs

En la siguiente sección se describe cómo ver y agregar informes desde CloudWatch Logs Insights a un panel personalizado de la consola de CloudWatch Logs.

### Visualización de los registros de una función

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Monitor (Monitorear).
4. Elija View logs in CloudWatch (Ver registros en CloudWatch).
5. Seleccione Ver en Logs Insights.
6. Elija entre los intervalos de tiempo predefinidos o elija un intervalo de tiempo personalizado.
7. Elija Ejecutar consulta.
8. (Opcional) Seleccione Guardar.

Select log group(s) ▼

Clear

/aws/lambda/wear\_heavy\_coat X

2020-05-01 (00:00:00) > 2020-12-31 (23:59:59) 📅

```

1  fields: @timestamp, @message
2  | sort @timestamp desc
3  | limit 20

```

Run query

Save

History

Logs
Visualization


Export results ▼

Add to dashboard

⚙️

Showing 20 of 144 records matched ⓘ Hide histogram

144 records (15.4 kB) scanned in 4.3s @ 33 records/s (3.6 kB/s)



The histogram shows a single vertical bar for the month of October, reaching a height of 33 on the y-axis. The x-axis is labeled with months from May to December.

#	@timestamp	@message
▶ 1	2020-09-29T18:54:16...	{'Weather': 'FREEZING'}

## Siguientes pasos

- Obtenga información sobre las métricas que Lambda registra y envía a CloudWatch en [Ver las métricas de funciones de Lambda](#).
- Aprenda a utilizar CloudWatch Lambda Insights para recopilar y agregar registros y métricas de rendimiento en tiempo de ejecución de la función de Lambda en [Supervise el rendimiento de las funciones con Amazon CloudWatch Lambda Insights](#).

# Ver las métricas de funciones de Lambda

Cuando la función de AWS Lambda termina de procesar un evento, Lambda envía métricas sobre la invocación a Amazon CloudWatch. No se aplica ningún cargo por estas métricas.

En la consola de CloudWatch, se pueden crear gráficos y paneles con estas métricas. Se pueden configurar alarmas para responder a cambios en el uso, el rendimiento o las tasas de error. Lambda envía datos de métricas a CloudWatch en intervalos de 1 minuto. Para obtener información más inmediata sobre una función de Lambda, puede crear [métricas personalizadas](#) de alta resolución como se describe en Serverless Land. Se aplican cargos por las métricas personalizadas y las alarmas de CloudWatch. Para obtener más información, consulte [Precios de Amazon CloudWatch](#).

Esta página describe las métricas de invocación de funciones, rendimiento y concurrencia de Lambda disponibles en la consola de CloudWatch.

## Secciones

- [Visualización de métricas en la consola de CloudWatch](#)
- [Tipos de métricas](#)

## Visualización de métricas en la consola de CloudWatch

Puede usar la consola CloudWatch para filtrar y clasificar métricas de funciones por nombre de función, alias o versión.

Para ver métricas en la consola de CloudWatch

1. Abra la [página Metrics \(Métricas\)](#) (espacio de nombres de AWS/Lambda) de la consola de CloudWatch.
2. En la pestaña Examinar, en Métricas, elija una de las siguientes dimensiones:
  - Por nombre de función (FunctionName): vea las métricas agregadas de todas las versiones y alias de una función.
  - Por recurso (Resource): vea las métricas de una versión o el alias de una función.
  - Por versión ejecutada (ExecutedVersion): vea las métricas para una combinación de alias y versión. Utilice la dimensión ExecutedVersion para comparar las tasas de error de dos versiones de una función que son objetivos de un [alias ponderado](#).



- En todas las funciones (ninguna): vea las métricas agregadas de todas las funciones de la Región de AWS actual.
3. Elija una métrica y, a continuación, seleccione Agregar al gráfico u otra opción del gráfico.

De forma predeterminada, los gráficos utilizan la estadística Sum para todas las métricas. Para elegir una estadística diferente y personalizar el gráfico, utilice las opciones de la pestaña Métricas Gráficas.

#### Note

La marca temporal de una métrica refleja cuándo se invocó la función. Dependiendo de la duración de la invocación, esto puede ser varios minutos antes de la emisión de la métrica. Por ejemplo, si la función tiene un tiempo de espera de 10 minutos, retroceda más de 10 minutos al buscar para obtener métricas precisas.

Para obtener más información acerca de CloudWatch, consulte la [Guía del usuario de Amazon CloudWatch](#).

## Tipos de métricas

En la siguiente sección, se describen los tipos de métricas de Lambda disponibles en la consola de CloudWatch.

### Métricas de invocación

Las métricas de invocación son indicadores binarios del resultado de la invocación de una función de Lambda. Por ejemplo, si la función devuelve un error, Lambda envía la métrica `Errors` con un valor de 1. Para obtener un recuento del número de errores de la función que se hayan producido cada minuto, consulte el valor de Sum de la métrica `Errors` con un período de 1 minuto.

#### Note

Consulte las siguientes métricas de invocación con la estadística Sum.

- `Invocations`: número de veces que se invoca el código de la función, incluidas las invocaciones correctas y las invocaciones que dan lugar a un error de la función. Las invocaciones no se

registran si la solicitud de invocación se limita o da lugar a un error de invocación. El valor de `Invocations` equivale al número de solicitudes facturadas.

- `Errors`: el número de invocaciones que dan lugar a un error de función. Los errores de la función incluyen las excepciones lanzadas por el código y las excepciones lanzadas por el tiempo de ejecución de Lambda. El motor de ejecución devuelve errores para problemas como tiempos de espera y errores de configuración. Para calcular la tasa de error, divida el valor de `Errors` por el valor de `Invocations`. Tenga en cuenta que la marca de hora de una métrica de error refleja cuándo se invocó la función, no cuando se produjo el error.
- `DeadLetterErrors`: en el caso de la [invocación asíncrona](#), el número de veces que Lambda intenta enviar un evento a una cola de mensajes fallidos (DLQ) sin conseguirlo. Pueden producirse errores de mensaje fallido debido a recursos mal configurados o límites de tamaño.
- `DestinationDeliveryFailures`: en el caso de la invocación asíncrona y la [asignación de orígenes de eventos](#) admitida, es el número de veces que Lambda intenta enviar un evento a un [destino](#) sin conseguirlo. Para la asignación de orígenes de eventos, Lambda admite los destinos de orígenes de secuencias (DynamoDB y Kinesis). Pueden producirse errores de entrega debido a errores de permisos, recursos mal configurados o límites de tamaño. También pueden producirse errores si el destino que ha configurado es de un tipo no admitido, como una cola FIFO de Amazon SQS o un tema FIFO de Amazon SNS.
- `Throttles`: el número de solicitudes de invocación que se han limitado. Cuando todas las instancias de función están procesando solicitudes y no hay simultaneidad disponible para escalar verticalmente, Lambda rechaza las solicitudes adicionales y muestra el error `TooManyRequestsException`. Las solicitudes limitadas y otros errores de invocación no cuentan como `Invocations` ni `Errors`.
- `OversizedRecordCount`: en el caso de los orígenes de eventos de Amazon DocumentDB, la cantidad de eventos que su función recibe de su flujo de cambios que tienen un tamaño superior a 6 MB. Lambda elimina el mensaje y emite esta métrica.
- `ProvisionedConcurrencyInvocations`: el número de veces que se invoca el código de la función con [simultaneidad aprovisionada](#).
- `ProvisionedConcurrencySpilloverInvocations`: el número de veces que se invoca el código de la función con simultaneidad estándar cuando ya se usa toda la simultaneidad aprovisionada.
- `RecursiveInvocationsDropped`: el número de veces que Lambda ha detenido la invocación de la función porque se detecta que la función forma parte de un bucle recursivo infinito. La [detección de bucles recursivos de Lambda](#) revisa la cantidad de veces que se invoca una función como parte de una cadena de solicitudes mediante el seguimiento de los metadatos agregados por

los AWS SDK compatibles. Si la función se invoca como parte de una cadena de solicitudes más de 16 veces, Lambda descarta la siguiente invocación.

## Métricas de desempeño

Las métricas de rendimiento proporcionan detalles de rendimiento sobre una única invocación de función. Por ejemplo, la métrica `Duration` indica la cantidad de tiempo en milisegundos que la función gasta procesando un evento. Para obtener una idea de la rapidez con la que su función procesa eventos, consulte estas métricas con la estadística `Average` o `Max`.

- `Duration`: la cantidad de tiempo que el código de función pasa procesando un evento. La duración facturada de una invocación es el valor de `Duration` redondeado hacia arriba a los milisegundos más cercanos. `Duration` no incluye el tiempo de arranque en frío.
- `PostRuntimeExtensionsDuration`: cantidad acumulativa de tiempo que el tiempo de ejecución dedica a ejecutar código para extensiones después de que se haya completado el código de función.
- `IteratorAge`: en el caso de los orígenes de eventos de DynamoDB, Kinesis y Amazon DocumentDB, la antigüedad del último registro del evento. Esta métrica mide el tiempo transcurrido desde que una secuencia recibe el registro hasta que la asignación de origen de eventos envía el evento a la función.
- `OffsetLag`: en el caso de los orígenes de eventos autoadministrados de Apache Kafka y Amazon Managed Streaming para Apache Kafka (Amazon MSK), la diferencia de desplazamiento entre el último registro escrito en un tema y el último registro que procesó el grupo de consumidores de la función. Aunque un tema de Kafka puede tener varias particiones, esta métrica mide el retraso de desplazamiento en el nivel del tema.

`Duration` también admite estadísticas percentiles ( $p$ ). Utilice percentiles para excluir valores atípicos que sesgan las estadísticas `Average` y `Maximum`. Por ejemplo, la estadística `p95` muestra la duración máxima del 95 % de las invocaciones, sin incluir el 5 % más lento. Para obtener más información, consulte [Percentiles](#) en la Guía del usuario de Amazon CloudWatch.

## Métricas de simultaneidad

Lambda informa de las métricas de simultaneidad como un recuento agregado del número de instancias que procesan eventos en una función, versión, alias o Región de AWS. Para ver lo cerca que está de alcanzar los [límites de simultaneidad](#), consulte estas métricas con la estadística `Max`.

- `ConcurrentExecutions`: número de instancias de función que están procesando eventos. Si este número alcanza la [cuota de ejecuciones simultáneas](#) de la región o el límite de [simultaneidad reservada](#) de la función, Lambda limita las solicitudes de invocación adicionales.
- `ProvisionedConcurrentExecutions`: el número de instancias de función que están procesando eventos con [simultaneidad aprovisionada](#). Para cada invocación de un alias o versión con simultaneidad aprovisionada, Lambda emite el recuento actual.
- `ProvisionedConcurrencyUtilization`: para una versión o alias, el valor de `ProvisionedConcurrentExecutions` dividido entre la cantidad total de simultaneidad aprovisionada asignada. Por ejemplo, si configura una simultaneidad aprovisionada de 10 para su función y su `ProvisionedConcurrentExecutions` es 7, entonces su `ProvisionedConcurrencyUtilization` es 0,7.
- `UnreservedConcurrentExecutions`: en el caso de región, número de eventos que están procesando las funciones que no tienen simultaneidad reservada.
- `ClaimedAccountConcurrency`: para una región, es la cantidad de simultaneidad que no está disponible para las invocaciones bajo demanda. `ClaimedAccountConcurrency` es igual a `UnreservedConcurrentExecutions` más la cantidad de simultaneidad asignada (es decir, la simultaneidad reservada total más la simultaneidad total aprovisionada). Para obtener más información, consulte [Cómo trabajar con la métrica `ClaimedAccountConcurrency`](#).

## Métricas de invocación asíncrona

Las métricas de invocación asíncrona proporcionan detalles sobre las invocaciones asíncronas desde los orígenes de eventos y las invocaciones directas. Puede establecer umbrales y alarmas para recibir notificaciones sobre ciertos cambios. Por ejemplo, cuando hay un aumento no deseado en el número de eventos en cola para su procesamiento (`AsyncEventsReceived`). O bien, cuando un evento ha estado esperando mucho tiempo para que se procese (`AsyncEventAge`).

- `AsyncEventsReceived`: el número de eventos que Lambda pone correctamente en cola para su procesamiento. Esta métrica proporciona información sobre la cantidad de eventos que recibe una función de Lambda. Supervise esta métrica y establezca alarmas para establecer umbrales para comprobar si hay problemas. Por ejemplo, para detectar un número no deseado de eventos enviados a Lambda y para diagnosticar rápidamente los problemas derivados de configuraciones incorrectas de desencadenadores o funciones. Los desajustes entre `AsyncEventsReceived` y `Invocations` pueden indicar una disparidad en el procesamiento, eventos que se descartan o un posible retraso en las colas.

- **AsyncEventAge**: el tiempo transcurrido entre el momento en que Lambda pone correctamente en cola el evento y el momento en que se invoca la función. El valor de esta métrica aumenta cuando se vuelven a intentar los eventos debido a errores de invocación o a limitaciones. Supervise esta métrica y establezca alarmas para conocer los umbrales de las diferentes estadísticas cuando se produzca una acumulación de colas. Para solucionar problemas de aumento en esta métrica, consulte la métrica **Errors** para identificar los errores de función y la métrica **Throttles** para identificar los problemas de simultaneidad.
- **AsyncEventsDropped**: el número de eventos que se descartan sin ejecutar correctamente la función. Si configura una cola de mensajes fallidos (DLQ) o un destino **OnFailure**, los eventos se enviarán allí antes de que se descarten. Los eventos se descartan por varias razones. Por ejemplo, los eventos pueden superar la antigüedad máxima del evento o agotar el número máximo de reintentos, o bien la simultaneidad reservada puede estar establecida en 0. Para solucionar problemas por los que se descartan los eventos, consulte la métrica **Errors** para identificar los errores de función y la métrica **Throttles** para identificar los problemas de simultaneidad.

# Uso de Registros de Amazon CloudWatch con AWS Lambda

AWS Lambda supervisa automáticamente funciones de Lambda en su nombre para ayudarlo a solucionar errores en sus funciones. Siempre que el [rol de ejecución](#) de la función cuente con los permisos necesarios, Lambda captura los registros de todas las solicitudes gestionadas por la función y los envía a Registros de Amazon CloudWatch.

Puede introducir instrucciones de registro en el código comprobar que el código está funcionando según lo previsto. Lambda se integra automáticamente a Registros de Amazon CloudWatch y envía todos los registros generados con el código a un grupo de registro de Amazon CloudWatch asociado a una función de Lambda.

De forma predeterminada, Lambda envía registros a un grupo de registro denominado `/aws/lambda/<function name>`. Si desea que su función envíe registros a otro grupo, puede configurar esta acción a través de la consola de Lambda, la AWS Command Line Interface (AWS CLI) o la API de Lambda. Consulte [the section called “Configuración de grupos de registros de CloudWatch”](#) para obtener más información.

Puede ver los registros de las funciones de Lambda mediante la consola de Lambda, la consola de CloudWatch, el AWS Command Line Interface (AWS CLI) o la API de CloudWatch.

## Note

Los registros pueden tardar de 5 a 10 minutos en aparecer después de una invocación de la función.

## Permisos de IAM necesarios

Su [rol de ejecución](#) necesita los siguientes permisos para cargar registros en los registros de CloudWatch:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Para obtener más información, consulte [Uso de políticas basadas en identidad \(políticas de IAM\) para los registros de CloudWatch](#) en la Guía del usuario de Amazon CloudWatch.

Puede agregar permisos de registros de CloudWatch mediante la política administrada de `AWSLambdaBasicExecutionRole` de AWS proporcionada por Lambda. Ejecute el siguiente comando para agregar esta política a su rol:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Para obtener más información, consulte [the section called “Políticas administradas de AWS”](#).

## Precios

No se aplican cargos adicionales por utilizar los registros de Lambda; no obstante, sí se aplican los cargos estándar de Registros de CloudWatch. Para obtener más información, consulte los [precios de CloudWatch](#).

## Configuración de controles de registro avanzados para las funciones de Lambda

Para tener más control sobre cómo se registran, procesan y consumen los registros de sus funciones, Lambda ofrece las siguientes opciones de configuración del registro:

- Formato de registro: seleccione entre texto sin formato y el formato JSON estructurado para los registros de su función
- Nivel de registro: para los registros estructurados en formato JSON, elija el nivel de detalle de los registros que Lambda envía a CloudWatch, como `ERROR`, `DEBUG` o `INFO`
- Grupo de registro: elija el grupo de registro de CloudWatch al que su función envía los registros

Para obtener más información sobre la configuración de los controles de registro avanzados, consulte las siguientes secciones:

### Temas

- [Configuración de los formatos de registro JSON y de texto sin formato](#)
- [Filtrado a nivel de registro](#)
- [Configuración de grupos de registros de CloudWatch](#)

## Configuración de los formatos de registro JSON y de texto sin formato

Al capturar las salidas del registro como pares clave-valor JSON, resulta más fácil buscar y filtrar resultados cuando se depuran las funciones. Con los registros con formato JSON, también puede agregar etiquetas e información contextual a sus registros. Esto puede ayudarlo a realizar un análisis automatizado de grandes volúmenes de datos de registro. A menos que su flujo de trabajo de desarrollo se base en herramientas existentes que consumen registros de Lambda con texto sin formato, le recomendamos que seleccione JSON como formato de registro.

Para todos los tiempos de ejecución administrados por Lambda, puede elegir si los registros del sistema de su función se deben enviar a Registros de Amazon CloudWatch en texto sin formato o formato JSON no estructurado. Los registros del sistema son los registros que Lambda genera y, a veces, se conocen como registros de eventos de plataforma.

En el caso de los [tiempos de ejecución compatibles](#), cuando se utiliza uno de los métodos de registro integrados compatibles, Lambda también puede generar los registros de aplicación de la función (los registros que genera el código de la función) en formato JSON estructurado. Al configurar el formato de registro de la función para estos tiempos de ejecución, la configuración que elija se aplicará tanto a los registros del sistema como de la aplicación.

Para los tiempos de ejecución compatibles, si su función utiliza una biblioteca o un método de registro compatibles, no necesita realizar ningún cambio en el código existente para que Lambda capture los registros en formato JSON estructurado.

### Note

El uso del formato de registro JSON agrega metadatos y codifica los mensajes de registro como objetos JSON que contienen una serie de pares clave-valor. Por este motivo, es posible que aumente el tamaño de los mensajes de registro de la función.

## Tiempos de ejecución y métodos de registro compatibles

Actualmente, Lambda admite la opción de generar registros de aplicaciones en formato JSON estructurado para los siguientes tiempos de ejecución.



Tiempo de ejecución	Versiones compatibles
Java	Todos los tiempos de ejecución de Java excepto Java 8 en Amazon Linux 1
Node.js	Node.js 16 y posteriores
Python	Python 3.8 y posteriores

Para que Lambda envíe los registros de aplicación de la función a CloudWatch en formato JSON estructurado, la función debe utilizar las siguientes herramientas de registro integradas para generar los registros:

- Java: el registrador `LambdaLogger` o `Log4j2`.
- Node.js: los métodos de la consola `console.trace`, `console.debug`, `console.log`, `console.info`, `console.error` y `console.warn`
- Python: la biblioteca `logging` de Python estándar

Para obtener más información sobre el uso de controles de registro avanzados con tiempos de ejecución compatibles, consulte [the section called “Registro”](#), [the section called “Registro”](#) y [the section called “Registro y supervisión de las funciones de Lambda de Python”](#).

Para otros tiempos de ejecución de Lambda administrados, Lambda actualmente solo admite de forma nativa la captura de registros del sistema en formato JSON estructurado. Sin embargo, puede seguir capturando los registros de aplicaciones en formato JSON estructurado en cualquier tiempo de ejecución con herramientas de registro, como Powertools para AWS Lambda, que generan salidas de registro con formato JSON.

### Formatos de registro predeterminados

Actualmente, el formato de registro predeterminado para todos los tiempos de ejecución de Lambda es texto sin formato.

Si ya utiliza bibliotecas de registro, como Powertools para AWS Lambda, para generar los registros de funciones en formato JSON estructurado, no necesita cambiar el código si selecciona el formato de registro JSON. Lambda no codifica dos veces ningún registro que ya esté codificado en JSON, por lo que los registros de la aplicación de su función seguirán capturándose como antes.

## Formato JSON para registros del sistema

Al configurar el formato de registro de la función como JSON, se captura cada elemento de registro del sistema (evento de plataforma) como un objeto JSON que contiene pares clave-valor con las siguientes claves:

- "time": la hora en que se generó el mensaje de registro
- "type": el tipo de evento que se está registrando
- "record": el contenido de la salida del registro

El formato del valor "record" varía según el tipo de evento que se registre. Para obtener más información, consulte [the section called “Tipos de objetos Event de la API de telemetría”](#). Para obtener más información sobre los niveles de registro asignados a los eventos de registro del sistema, consulte [the section called “Asignación de eventos a nivel de registro del sistema”](#).


A modo de comparación, los dos ejemplos siguientes muestran la misma salida de registro en formato JSON estructurado y de texto sin formato. Tenga en cuenta que, en la mayoría de los casos, los eventos de registro del sistema contienen más información cuando se generan con formato JSON que cuando se generan con texto sin formato.

Example Texto sin formato:

```
2023-03-13 18:56:24.046000 fbe8c1 INIT_START Runtime Version:
python:3.9.v18 Runtime Version ARN: arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0
```

Example JSON estructurado:

```
{
  "time": "2023-03-13T18:56:24.046Z",
  "type": "platform.initStart",
  "record": {
    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "python:3.9.v18",
    "runtimeVersionArn": "arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0"
  }
}
```

 Note


La [the section called “API de telemetría”](#) siempre emite eventos de plataforma, como START y REPORT, en formato JSON. La configuración del formato de los registros del sistema que Lambda envía a CloudWatch no afecta el comportamiento de la API de telemetría de Lambda.

## Formato JSON para registros de aplicaciones

Al configurar el formato de registro de la función como JSON, las salidas del registro de aplicación escritas con las bibliotecas y los métodos de registro compatibles se capturan como un objeto JSON que contiene pares de clave-valor con las siguientes claves:

- "timestamp": la hora en que se generó el mensaje de registro
- "level": el nivel de registro asignado al mensaje
- "message": el contenido del mensaje de registro
- "requestId" (Python y Node.js) o "AWSrequestId" (Java): el ID de solicitud único para la invocación de la función

Según el tiempo de ejecución y el método de registro que utilice la función, este objeto JSON también puede contener pares de claves adicionales. Por ejemplo, en Node.js, si la función utiliza métodos `console` para registrar objetos de error con varios argumentos, el objeto JSON contendrá pares de clave-valor adicionales junto con las claves `errorMessage`, `errorType`, y `ystackTrace`. Para obtener más información sobre los registros con formato JSON en diferentes tiempos de ejecución de Lambda, consulte [the section called “Registro y supervisión de las funciones de Lambda de Python”](#), [the section called “Registro”](#), y [the section called “Registro”](#).

 Note

La clave que Lambda utiliza para el valor de marca de tiempo es diferente para los registros del sistema y los registros de la aplicación. En el caso de los registros del sistema, Lambda usa la clave "time" para mantener la coherencia con la API de telemetría. Para los registros de la aplicación, Lambda sigue las convenciones de los tiempos de ejecución compatibles y utiliza "timestamp".

A modo de comparación, los dos ejemplos siguientes muestran la misma salida de registro en formato JSON estructurado y de texto sin formato.

Example Texto sin formato:

```
2023-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON estructurado:

```
{
  "timestamp": "2023-10-27T19:17:45.586Z",
  "level": "INFO",
  "message": "some log message",
  "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

## Configuración del formato de registro de su función

Para configurar el formato de registro de la función, puede utilizar la consola de Lambda o la AWS Command Line Interface (AWS CLI). También puede configurar el formato de registro de una función mediante los comandos de la API de Lambda [CreateFunction](#) y [UpdateFunctionConfiguration](#), el recurso de AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#) y el recurso de AWS CloudFormation [AWS::Lambda::Function](#).

Cambiar el formato de registro de la función no afecta a los registros existentes almacenados en Registros de Amazon CloudWatch. Solo los registros nuevos utilizarán el formato actualizado.

Si cambia el formato de registro de la función a JSON y no establece el nivel de registro, Lambda establece automáticamente el nivel de registro de la aplicación y el nivel de registro del sistema de la función en INFO. Esto significa que Lambda envía a Registros de CloudWatch solo las salidas de registro de nivel INFO e inferiores. Para obtener más información sobre el filtrado a nivel de registro de aplicaciones y sistemas, consulte [the section called "Filtrado a nivel de registro"](#).

### Note

Para los tiempos de ejecución de Python, cuando el formato de registro de la función está establecido en texto sin formato, la configuración de nivel de registro predeterminada es WARN. Esto significa que Lambda envía a Registros de CloudWatch solo las salidas de registro de nivel WARN e inferiores. Al cambiar el formato de registro de la función a JSON, se modifica este comportamiento predeterminado. Para obtener más información acerca de

los registros de Python, consulte [the section called “Registro y supervisión de las funciones de Lambda de Python”](#).

En el caso de las funciones de Node.js que emiten registros en formato de métricas integradas (EMF), cambiar el formato de registro de la función a JSON podría provocar que CloudWatch no reconozca las métricas.

#### Important

Si su función utiliza Powertools para AWS Lambda (TypeScript) o las bibliotecas cliente de EMF de código abierto para emitir registros EMF, actualice las bibliotecas de [Powertools](#) y [EMF](#) a las versiones más recientes para asegurarse de que CloudWatch pueda seguir analizando los registros correctamente. Si cambia al formato de registro JSON, también le recomendamos que realice pruebas para asegurarse de que es compatible con las métricas integradas de la función. Para obtener más información sobre las funciones de node.js que emiten registros EMF, consulte [the section called “Uso de bibliotecas cliente de formato de métricas integradas \(EMF\) con registros JSON estructurados”](#).

Para configurar el formato de registro de una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En la página de configuración de funciones, elija Herramientas de supervisión y operaciones.
4. En el panel Configuración de registros, seleccione Editar.
5. En Contenido del registro, para Formato de registro, seleccione Texto o JSON.
6. Seleccione Guardar.

Cambio del formato de registro de una función existente (AWS CLI)

- Para cambiar el formato de registro de una función existente, utilice el comando `update-function-configuration`. Configure la opción `LogFormat` en `LoggingConfig` para que sea `JSON` o `Text`.

```
aws lambda update-function-configuration \
```

```
--function-name myFunction --logging-config LogFormat=JSON
```

Para establecer el formato de registro al crear una función (AWS CLI)

- Para configurar el formato de registro al crear una función nueva, utilice la opción `--logging-config` en el comando `create-function`. Establezca `LogFormat` en `JSON` o `Text`. El siguiente comando de ejemplo crea una función con el tiempo de ejecución Node.js 18 que genera registros en formato JSON estructurado.

Si no especifica un formato de registro al crear una función, Lambda utilizará el formato de registro predeterminado para la versión de tiempo de ejecución que seleccione. Para obtener información sobre los formatos de registro predeterminados, consulte [the section called “Formatos de registro predeterminados”](#).

```
aws lambda create-function --function-name myFunction --runtime nodejs18.x \  
--handler index.handler --zip-file fileb://function.zip \  
--role arn:aws:iam::123456789012:role/LambdaRole --logging-config LogFormat=JSON
```

## Filtrado a nivel de registro

Lambda puede filtrar los registros de su función para que solo se envíen a Registros de Amazon CloudWatch los registros con un nivel de detalle determinado o inferior. Puede configurar el filtrado a nivel de registro por separado para los registros del sistema de la función (los registros que Lambda genera) y los registros de la aplicación (los registros que genera el código de la función).

Para [the section called “Tiempos de ejecución y métodos de registro compatibles”](#), no necesita realizar ningún cambio en el código de la función para que Lambda filtre los registros de aplicación de su función.

Para todos los demás tiempos de ejecución y métodos de registro, el código de la función debe generar los eventos de registro en `stdout` o `stderr` como objetos con formato JSON que contengan un par clave-valor con la clave `level`. Por ejemplo, Lambda interpreta la siguiente salida de `stdout` como un registro de nivel `DEBUG`.

```
print({'level': "debug", "msg": "my debug log", "timestamp":  
"2023-11-02T16:51:31.587199Z"})
```

Si el campo de valor "level" no es válido o falta, Lambda asignará a la salida del registro el nivel INFO. Para que Lambda utilice el campo de marca de tiempo, debe especificar el tiempo con un formato de marca de tiempo [RFC 3339](#) válido. Si no proporciona una marca de tiempo válida, Lambda asignará al registro el nivel INFO y agregará una marca de tiempo por usted.

Cuando asigne un nombre a la clave de marca de tiempo, siga las convenciones del tiempo de ejecución que esté utilizando. Lambda admite la mayoría de las convenciones de nomenclatura habituales que utilizan los tiempos de ejecución administrados. Por ejemplo, en las funciones que utilizan el tiempo de ejecución .NET, Lambda reconoce la clave "Timestamp".

### Note

Para usar el filtrado a nivel de registro, la función debe estar configurada para usar el formato de registro JSON. Actualmente, el formato de registro predeterminado para todos los tiempos de ejecución administrados por Lambda es texto sin formato. Para obtener información sobre cómo configurar el formato de registro de la función como JSON, consulte [the section called "Configuración del formato de registro de su función"](#).

Para los registros de aplicaciones (los registros generados por el código de la función), puede elegir entre los siguientes niveles de registro.

Nivel de registro	Uso estándar
TRACE (más detallado)	La información más detallada que se utiliza para rastrear la ruta de ejecución del código
DEBUG	Información detallada para la depuración del sistema
INFO	Mensajes que registran el funcionamiento normal de su función
WARN	Mensajes sobre posibles errores que pueden provocar un comportamiento inesperado si no se abordan
ERROR	Mensajes sobre problemas que impiden que el código funcione según lo esperado

Nivel de registro	Uso estándar
FATAL (menos detallado)	Mensajes sobre errores graves que hacen que la aplicación deje de funcionar

Al seleccionar un nivel de registro, Lambda envía los registros de ese nivel y de niveles inferiores a Registros de Amazon CloudWatch. Por ejemplo, si establece el nivel de registro de la aplicación de una función en WARN, Lambda no envía las salidas de registro en los niveles INFO y DEBUG. El nivel de registro de aplicaciones predeterminado para el filtrado de registros es INFO.

Cuando Lambda filtra los registros de aplicaciones de la función, a los mensajes de registro sin nivel se les asignará el nivel de registro INFO.

Para los registros del sistema (los registros generados por el servicio Lambda), puede elegir entre los siguientes niveles de registro.

Nivel de registro	Uso
DEBUG (más detallado)	Información detallada para la depuración del sistema
INFO	Mensajes que registran el funcionamiento normal de su función
WARN (menos detallado)	Mensajes sobre posibles errores que pueden provocar un comportamiento inesperado si no se abordan

Al seleccionar un nivel de registro, Lambda envía los registros a ese nivel y a niveles inferiores. Por ejemplo, si establece el nivel de registro del sistema de una función en INFO, Lambda no envía las salidas de registro en el nivel DEBUG.

De forma predeterminada, Lambda establece el nivel de registro del sistema en INFO. Con esta configuración, Lambda envía los mensajes de registro "start" y "report" a CloudWatch automáticamente. Para recibir registros del sistema más o menos detallados, cambie el nivel de registro a DEBUG o WARN. Para ver una lista de los niveles de registro a los que Lambda asigna



diferentes eventos de registro del sistema, consulte [the section called “Asignación de eventos a nivel de registro del sistema”](#).

## Configuración del filtrado a nivel de registro

Para configurar el filtrado a nivel de registro de aplicaciones y sistemas para su función, puede utilizar la consola de Lambda o la AWS Command Line Interface (AWS CLI). También puede configurar el nivel de registro de una función con los comandos de la API de Lambda [CreateFunction](#) y [UpdateFunctionConfiguration](#), el recurso de AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#) y el recurso de AWS CloudFormation [AWS::Lambda::Function](#).

Tenga en cuenta que si establece el nivel de registro de la función en el código, esta configuración tiene prioridad sobre cualquier otra configuración a nivel de registro que configure. Por ejemplo, si usa el método `setLevel()` de logging de Python para establecer el nivel de registro de la función en INFO, esta configuración tiene prioridad sobre la configuración WARN que configure mediante la consola de Lambda.

Para configurar el nivel de registro de sistema o aplicación de una función existente (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En la página de configuración de funciones, elija Herramientas de supervisión y operaciones.
4. En el panel Configuración de registros, seleccione Editar.
5. En Contenido del registro, para Formato de registro, asegúrese de seleccionar JSON.
6. Con los botones de opción, seleccione el Nivel de registro de la aplicación y el Nivel de registro del sistema que desee para su función.
7. Seleccione Guardar.

Para configurar el nivel de registro de la aplicación o del sistema de una función existente (AWS CLI)

- Para cambiar el nivel de registro de la aplicación o del sistema de una función existente, utilice el comando `update-function-configuration`. Configure `--system-log-level` en una de las siguientes opciones: DEBUG, INFO o WARN. Configure `--application-log-level` en una de las siguientes opciones: DEBUG, INFO, WARN, ERROR o FATAL.

```
aws lambda update-function-configuration \  
--function-name myFunction --system-log-level WARN \  

```

**--application-log-level ERROR**

Para configurar el filtrado a nivel de registro al crear una función

- Para configurar el filtrado a nivel de registro al crear una función nueva, utilice las opciones `--system-log-level` y `--application-log-level` en el comando `create-function`. Configure `--system-log-level` en una de las siguientes opciones: `DEBUG`, `INFO` o `WARN`. Configure `--application-log-level` en una de las siguientes opciones: `DEBUG`, `INFO`, `WARN`, `WARN` o `FATAL`.

```
aws lambda create-function --function-name myFunction --runtime nodejs18.x \
--handler index.handler --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/LambdaRole --system-log-level WARN \
--application-log-level ERROR
```

Asignación de eventos a nivel de registro del sistema

Para los eventos de registro a nivel de sistema generados por Lambda, la siguiente tabla define el nivel de registro asignado a cada evento. Para obtener más información sobre los eventos que figuran en la tabla, consulte [the section called “Referencia del esquema Event”](#).

Nombre de evento	Condición	Nivel de registro asignado
initStart	runtimeVersion está configurado	INFO
initStart	runtimeVersion no está configurado	DEBUG
initRuntimeDone	status=success	DEBUG
initRuntimeDone	status!=success	WARN
initReport	initializationType=snapstart	INFO
initReport	initializationType!=snapstart	DEBUG
initReport	status!=success	WARN

Nombre de evento	Condición	Nivel de registro asignado
restoreStart	runtimeVersion está configurado	INFO
restoreStart	runtimeVersion no está configurado	DEBUG
restoreRuntimeDone	status=success	DEBUG
restoreRuntimeDone	status!=success	WARN
restoreReport	status=success	INFO
restoreReport	status!=success	WARN
iniciar	-	INFO
runtimeDone	status=success	DEBUG
runtimeDone	status!=success	WARN
report	status=success	INFO
report	status!=success	WARN
extensión	state=success	INFO
extensión	state!=success	WARN
logSubscription	-	INFO
telemetrySubscription	-	INFO
logsDropped	-	WARN

**Note**

La [the section called “API de telemetría”](#) siempre emite el conjunto completo de eventos de plataforma. La configuración del nivel de los registros del sistema que Lambda envía a CloudWatch no afecta el comportamiento de la API de telemetría de Lambda.

## Filtrado a nivel de registro de aplicaciones con tiempos de ejecución personalizados

Al configurar el filtrado a nivel de registro de aplicaciones para su función, en segundo plano, Lambda establece el nivel de registro de la aplicación en el tiempo de ejecución mediante la variable de entorno `AWS_LAMBDA_LOG_LEVEL`. Lambda también establece el formato de registro de la función mediante la variable de entorno `AWS_LAMBDA_LOG_FORMAT`. Puede utilizar estas variables para integrar los controles de registro avanzados de Lambda en un [tiempo de ejecución personalizado](#).

Para poder configurar los ajustes de registro de una función mediante un tiempo de ejecución personalizado con la consola de Lambda, AWS CLI y las API de Lambda, configure el tiempo de ejecución personalizado para comprobar el valor de estas variables de entorno. A continuación, puede configurar los registradores de su tiempo de ejecución de acuerdo con el formato y los niveles de registro que seleccione.

## Configuración de grupos de registros de CloudWatch

De forma predeterminada, CloudWatch crea automáticamente un grupo de registro denominado `/aws/lambda/<function name>` para la función cuando se invoca por primera vez. Para configurar su función de manera que envíe registros a un grupo de registro existente o para crear un nuevo grupo de registro para su función, puede usar la consola de Lambda o la AWS CLI. También puede configurar grupos de registros personalizados con los comandos de la API de Lambda [CreateFunction](#) y [UpdateFunctionConfiguration](#), y el recurso de AWS Serverless Application Model (AWS SAM) [AWS::Serverless::Function](#).

Puede configurar varias funciones de Lambda para enviar registros al mismo grupo de registro de CloudWatch. Por ejemplo, puede usar un único grupo de registro para almacenar los registros de todas las funciones de Lambda que componen una aplicación concreta. Cuando se utiliza un grupo de registro personalizado para una función de Lambda, los flujos de registro que Lambda crea incluyen el nombre y la versión de la función. Esto garantiza que se conserve la asignación entre los mensajes de registro y las funciones, incluso si utiliza el mismo grupo de registro para varias funciones.

El formato de denominación del flujo de registro para grupos de registro personalizados sigue esta convención:

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Tenga en cuenta que, al configurar un grupo de registro personalizado, el nombre que seleccione para el grupo de registro debe seguir las [reglas de nomenclatura de Registros de Amazon CloudWatch](#). Además, los nombres personalizados de los grupos de registros no deben empezar por la cadena `aws/`. Si crea un grupo de registros personalizado empezando por `aws/`, Lambda no podrá crear el grupo de registros. Como resultado, los registros de la función no se enviarán a CloudWatch.

#### Cambio del grupo de registro de una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. En la página de configuración de funciones, elija Herramientas de supervisión y operaciones.
4. En el panel Configuración de registros, seleccione Editar.
5. En el panel Grupo de registro, para el Grupo de registro de CloudWatch, elija Personalizado.
6. En Grupo de registro personalizado, introduzca el nombre del grupo de registro de CloudWatch al que quiere que su función envíe registros. Si introduce el nombre de un grupo de registro existente, su función utilizará ese grupo. Si no existe ningún grupo de registro con el nombre que ingresa, Lambda creará un nuevo grupo de registro para su función con ese nombre.

#### Cambio del grupo de registro de una función (AWS CLI)

- Para cambiar el grupo de registro de una función existente, utilice el comando `update-function-configuration`. Si especifica el nombre de un grupo de registro existente, la función utilizará ese grupo. Si no existe ningún grupo de registro con el nombre que especifica, Lambda creará un nuevo grupo de registro para su función con ese nombre.

```
aws lambda update-function-configuration \  
--function-name myFunction --log-group myLogGroup
```

## Para especificar un grupo de registro personalizado al crear una función (AWS CLI)

- Para especificar un grupo de registro personalizado al crear una nueva función de Lambda con la AWS CLI, utilice la opción `--log-group`. Si especifica el nombre de un grupo de registro existente, la función utilizará ese grupo. Si no existe ningún grupo de registro con el nombre que especifica, Lambda creará un nuevo grupo de registro para su función con ese nombre.

El siguiente comando de ejemplo crea una función de Lambda de Node.js que envía registros a un grupo de registro denominado `myLogGroup`.

```
aws lambda create-function --function-name myFunction --runtime nodejs18.x \  
--handler index.handler --zip-file fileb://function.zip \  
--role arn:aws:iam::123456789012:role/LambdaRole --log-group myLogGroup
```

## Permisos de rol de ejecución

Para que su función envíe registros a Registros de Amazon CloudWatch, debe tener el permiso [logs:PutLogEvents](#). Al configurar el grupo de registro de la función con la consola de Lambda, si la función no tiene este permiso, Lambda lo agrega al [rol de ejecución](#) de la función de forma predeterminada. Cuando Lambda agrega este permiso, otorga a la función permiso para enviar registros a cualquier grupo de registro de Registros de Amazon CloudWatch.

Para evitar que Lambda actualice automáticamente el rol de ejecución de la función y, en su lugar, lo edite manualmente, expanda Permisos y deje la opción Agregar permisos necesarios sin marcar.

Al configurar el grupo de registro de la función con la AWS CLI, Lambda no agregará el permiso `logs:PutLogEvents` automáticamente. Agregue el permiso al rol de ejecución de su función si aún no lo tiene. Este permiso se incluye en la política administrada [AWSLambdaBasicExecutionRole](#).

## Visualización de los registros de función de Lambda de Amazon CloudWatch

Puede ver los registros de CloudWatch para la función de Lambda mediante la consola de los registros de CloudWatch o la AWS Command Line Interface (AWS CLI). Siga las instrucciones en las siguientes secciones para acceder a los registros de la función.

## Acceso a los registros mediante la consola

Para visualizar los registros mediante la consola de CloudWatch haga lo siguiente:

1. En la consola de CloudWatch, abra la [página de grupos de registro](#)
2. Seleccione el grupo de registro de la función. El formato de nombre predeterminado para los grupos de registro de CloudWatch es `/aws/lambda/<function_name>`.
3. Seleccione el flujo de registro para las invocaciones de la función que desee consultar.

También puede acceder a los registros de CloudWatch de la función directamente desde la consola de Lambda.

Visualización de los registros mediante la consola de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Seleccione una función.
3. Elegir Monitor.
4. Elija View logs in CloudWatch (Ver registros en CloudWatch).
5. Seleccione el flujo de registro para las invocaciones de la función que desee consultar.

## Acceso a los registros con la AWS CLI

La AWS CLI es una herramienta de código abierto que lo habilita para interactuar con los servicios de AWS mediante el uso de comandos en el shell de la línea de comandos. Para completar los pasos de esta sección, debe disponer de lo siguiente:

- [AWS Command Line Interface \(AWS CLI\) versión 2](#)
- [AWS CLI: configuración rápida con `aws configure`](#)

Puede utilizar la [CLI de AWS CLI](#) para recuperar registros de una invocación mediante la opción de comando `--log-type`. La respuesta contiene un campo `LogResult` que contiene hasta 4 KB de registros con codificación base64 a partir de la invocación.

Example recuperar un ID de registro

En el ejemplo siguiente se muestra cómo recuperar un ID de registro del campo `LogResult` para una función denominada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Debería ver los siguientes datos de salida:

```
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElk0iA4N2QwNDRi0C1mMTU0LTEXZTgt0GNkYS0y0Tc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

Example decodificar los registros

En el mismo símbolo del sistema, utilice la utilidad base64 para decodificar los registros. En el ejemplo siguiente se muestra cómo recuperar registros codificados en base64 para my-function.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

La opción cli-binary-format es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

Debería ver los siguientes datos de salida:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ22luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0\"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Duration: 79.67 ms Billed
Duration: 80 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

La utilidad base64 está disponible en Linux, macOS y [Ubuntu en Windows](#). Es posible que los usuarios de macOS necesiten usar `base64 -D`.



## Example get-logs.sh script

En el mismo símbolo del sistema, utilice el siguiente script para descargar los últimos cinco eventos de registro. El script utiliza sed para eliminar las comillas del archivo de salida y permanece inactivo durante 15 segundos para dar tiempo a que los registros estén disponibles. La salida incluye la respuesta de Lambda y la salida del comando `get-log-events`.

Copie el contenido de la siguiente muestra de código y guárdelo en su directorio de proyecto Lambda como `get-logs.sh`.

La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la AWS CLI. Para que esta sea la configuración predeterminada, ejecute `aws configure set cli-binary-format raw-in-base64-out`. Para obtener más información, consulte [Opciones de la línea de comandos globales compatibles con AWS CLI](#) en la Guía del usuario de la AWS Command Line Interface versión 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

## Example macOS y Linux (solamente)

En el mismo símbolo del sistema, es posible que los usuarios de macOS y Linux necesiten ejecutar el siguiente comando para asegurarse de que el script es ejecutable.

```
chmod -R 755 get-logs.sh
```

## Example recuperar los últimos cinco eventos de registro

En el mismo símbolo del sistema, ejecute el siguiente script para obtener los últimos cinco eventos de registro.

```
./get-logs.sh
```

Debería ver los siguientes datos de salida:

```

{
  "statusCode": 200,
  "executedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"\n$LATEST\n",
\r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003218,
      "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

# Registrar llamadas a la API de AWS Lambda mediante AWS CloudTrail

AWS Lambda se integra con [AWS CloudTrail](#), un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un Servicio de AWS. CloudTrail captura las llamadas a la API de Lambda como eventos. Las llamadas capturadas incluyen las llamadas desde la consola de Lambda y las llamadas desde el código a las operaciones de la API de Lambda. Mediante la información recopilada por CloudTrail, puede determinar la solicitud que se realizó a Lambda, la dirección IP desde la que se realizó, cuándo se realizó y detalles adicionales.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales del usuario raíz o del usuario.
- Si la solicitud se realizó en nombre de un usuario de IAM Identity Center.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro Servicio de AWS.

CloudTrail está activado en la Cuenta de AWS cuando usted crea la cuenta y tiene acceso automático al Historial de eventos de CloudTrail. El Historial de eventos de CloudTrail proporciona un registro visible e inmutable, que se puede buscar y descargar, de los últimos 90 días de eventos de gestión registrados en una Región de AWS. Para obtener más información, consulte [Trabajar con el historial de eventos de CloudTrail](#) en la Guía del usuario de AWS CloudTrail. No se cobran cargos de CloudTrail por ver el Historial de eventos.

Para mantener un registro permanente de los eventos en su Cuenta de AWS más allá de los 90 días, cree un registro de seguimiento o un almacén de datos de eventos de [CloudTrail Lake](#).

## Registros de seguimiento de CloudTrail

Un registro de seguimiento permite a CloudTrail enviar archivos de registro a un bucket de Amazon S3. Todos los registros de seguimiento que cree con la AWS Management Console son de varias regiones. Puede crear un registro de seguimiento de una sola región o de varias regiones mediante la AWS CLI. Se recomienda crear un registro de seguimiento de varias regiones, ya que registra actividad en todas las Regiones de AWS de su cuenta. Si crea un registro de seguimiento de una sola región, solo podrá ver los eventos registrados en la Región

de AWS del registro de seguimiento. Para obtener más información acerca de los registros de seguimiento, consulte [Creación de un registro de seguimiento para su Cuenta de AWS](#) y [Creación de un registro de seguimiento para una organización](#) en la Guía del usuario de AWS CloudTrail.

Puede crear un registro de seguimiento para enviar una copia de los eventos de administración en curso en su bucket de Amazon S3 sin costo alguno desde CloudTrail; sin embargo, hay cargos por almacenamiento en Amazon S3. Para obtener más información sobre los precios de CloudTrail, consulte [Precios de AWS CloudTrail](#). Para obtener información acerca de los precios de Amazon S3, consulte [Precios de Amazon S3](#).

## Almacenes de datos de eventos de CloudTrail Lake

CloudTrail Lake le permite ejecutar consultas basadas en SQL sobre los eventos. CloudTrail Lake convierte los eventos existentes en formato JSON basado en filas al formato [ORC de Apache](#). ORC es un formato de almacenamiento en columnas optimizado para una recuperación rápida de datos. Los eventos se agregan en almacenes de datos de eventos, que son recopilaciones inmutables de eventos en función de criterios que se seleccionan aplicando [selectores de eventos avanzados](#). Los selectores que se aplican a un almacén de datos de eventos controlan los eventos que perduran y están disponibles para la consulta. Para obtener más información acerca de CloudTrail Lake, consulte [Trabajar con AWS CloudTrail Lake](#) en la Guía del usuario de AWS CloudTrail.

Los almacenes de datos de eventos de CloudTrail Lake y las consultas generan costos adicionales. Cuando crea un almacén de datos de eventos, elige la [opción de precios](#) que desea utilizar para él. La opción de precios determina el costo de la incorporación y el almacenamiento de los eventos, así como el periodo de retención predeterminado y máximo del almacén de datos de eventos. Para obtener más información sobre los precios de CloudTrail, consulte [Precios de AWS CloudTrail](#).

## Eventos de datos de Lambda en CloudTrail

Los [eventos de datos](#) proporcionan información sobre las operaciones de recursos realizadas en o dentro de un recurso (por ejemplo, leer o escribir en un objeto de Amazon S3). Se denominan también operaciones del plano de datos. Los eventos de datos suelen ser actividades de gran volumen. De forma predeterminada, CloudTrail no registra la mayoría de los eventos de datos y el historial de eventos de CloudTrail no los registra.

Un evento de datos de CloudTrail que se registra de forma predeterminada para los servicios compatibles es `LambdaESMDisabled`. Para obtener más información sobre el uso de este evento

para solucionar problemas con las asignaciones de orígenes de eventos de Lambda, consulte [the section called “Uso de CloudTrail para solucionar problemas de orígenes de eventos de Lambda deshabilitados”](#).

Se aplican cargos adicionales a los eventos de datos. Para obtener más información sobre los precios de CloudTrail, consulte [Precios de AWS CloudTrail](#).

Puede registrar eventos de datos para el tipo de recurso de `AWS::Lambda::Function` mediante la consola de CloudTrail, la AWS CLI o las operaciones de la API de CloudTrail. Para obtener más información sobre cómo registrar los eventos de datos, consulte [Registro de eventos de datos con la AWS Management Console](#) y [Registro de eventos de datos con la AWS Command Line Interface](#) en la Guía del usuario de AWS CloudTrail.

En la siguiente tabla se muestra el tipo de recurso Lambda para el que puede registrar eventos de datos. La columna Tipo de evento de datos (consola) muestra el valor que se debe elegir en la lista de tipos de eventos de datos de la consola de CloudTrail. La columna `resources.type value` muestra el valor de `resources.type`, que especificaría al configurar los selectores de eventos avanzados mediante la AWS CLI o las API de CloudTrail. La columna API de datos registradas en CloudTrail muestra las llamadas a la API registradas en CloudTrail para el tipo de recurso.

Tipo de evento de datos (consola)	<code>resources.type value</code>	API de datos registradas en CloudTrail
Lambda	<code>AWS::Lambda::Function</code>	<a href="#">Invoke</a>

Puede configurar selectores de eventos avanzados para filtrar según los campos `eventName`, `readOnly` y `resources.ARN` y así registrar solo los eventos que son importantes para usted. El siguiente ejemplo es la vista JSON de una configuración de eventos de datos que registra eventos solo para una función específica. Para obtener más información acerca de estos campos, consulte [AdvancedFieldSelector](#) en la Referencia de la API de AWS CloudTrail.

```
[
  {
    "name": "function-invokes",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
```

```

    "Data"
  ]
},
{
  "field": "resources.type",
  "equals": [
    "AWS::Lambda::Function"
  ]
},
{
  "field": "resources.ARN",
  "equals": [
    "arn:aws:lambda:us-east-1:111122223333:function:hello-world"
  ]
}
]
}
]

```

## Eventos de administración de Lambda en CloudTrail

Los [eventos de administración](#) proporcionan información sobre las operaciones de administración que se realizan en los recursos de su Cuenta de AWS. Se denominan también operaciones del plano de control. CloudTrail registra los eventos de administración de forma predeterminada.

Lambda admite el registro de las siguientes acciones como eventos de administración en los archivos de registro de CloudTrail.

### Note

En el archivo de registro de CloudTrail, el `eventName` puede incluir información de la fecha y la versión, pero sigue haciendo referencia a la misma acción pública de la API. Por ejemplo, la acción `GetFunction` aparece como `GetFunction20150331v2`. La siguiente lista especifica en qué casos el nombre del evento difiere del nombre de la acción de la API.

- [AddLayerVersionPermission](#)
- [AddPermission](#) (nombre del evento: `AddPermission20150331v2`)
- [CreateAlias](#) (nombre del evento: `CreateAlias20150331`)
- [CreateEventSourceMapping](#) (nombre del evento: `CreateEventSourceMapping20150331`)

- [CreateFunction](#) (nombre del evento: CreateFunction20150331)

(Se omiten los parámetros Environment y ZipFile de los registros de CloudTrail para CreateFunction).

- [CreateFunctionUrlConfig](#)
- [DeleteAlias](#) (nombre del evento: DeleteAlias20150331)
- [DeleteCodeSigningConfig](#)
- [DeleteEventSourceMapping](#) (nombre del evento: DeleteEventSourceMapping20150331)
- [DeleteFunction](#) (nombre del evento: DeleteFunction20150331)
- [DeleteFunctionConcurrency](#) (nombre del evento: DeleteFunctionConcurrency20171031)
- [DeleteFunctionUrlConfig](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [GetAlias](#) (nombre del evento: GetAlias20150331)
- [GetEventSourceMapping](#)
- [GetFunction](#)
- [GetFunctionUrlConfig](#)
- [GetFunctionConfiguration](#)
- [GetLayerVersionPolicy](#)
- [GetPolicy](#)
- [ListEventSourceMappings](#)
- [ListFunctions](#)
- [ListFunctionUrlConfigs](#)
- [PublishLayerVersion](#) (nombre del evento: PublishLayerVersion20181031)

(Se omita el parámetro ZipFile de los registros de CloudTrail para PublishLayerVersion).

- [PublishVersion](#) (nombre del evento: PublishVersion20150331)
- [PutFunctionConcurrency](#) (nombre del evento: PutFunctionConcurrency20171031)
- [PutFunctionCodeSigningConfig](#)
- [PutFunctionEventInvokeConfig](#)
- [PutProvisionedConcurrencyConfig](#)
- [PutRuntimeManagementConfig](#)
- [RemovePermission](#) (nombre del evento: RemovePermission20150331v2)

- [TagResource](#) (nombre del evento: TagResource20170331v2)
- [UntagResource](#) (nombre del evento: UntagResource20170331v2)
- [UpdateAlias](#) (nombre del evento: UpdateAlias20150331)
- [UpdateCodeSigningConfig](#)
- [UpdateEventSourceMapping](#) (nombre del evento: UpdateEventSourceMapping20150331)
- [UpdateFunctionCode](#) (nombre del evento: UpdateFunctionCode20150331v2)

(Se omite el parámetro ZipFile de los registros de CloudTrail para UpdateFunctionCode).

- [UpdateFunctionConfiguration](#) (nombre del evento: UpdateFunctionConfiguration20150331v2)

(Se omite el parámetro Environment de los registros de CloudTrail para UpdateFunctionConfiguration).

- [UpdateFunctionEventInvokeConfig](#)
- [UpdateFunctionUrlConfig](#)

## Uso de CloudTrail para solucionar problemas de orígenes de eventos de Lambda deshabilitados

Cuando cambia el estado de la asignación de orígenes de eventos mediante la acción de la API [UpdateEventSourceMapping](#), la llamada a la API se registra como un evento de administración en CloudTrail. Las asignaciones de orígenes de eventos también pueden pasar directamente al estado Disabled debido a errores.

Para los siguientes servicios, Lambda publica el evento de datos LambdaESMDisabled en CloudTrail cuando el origen del evento pasa al estado Deshabilitado:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda no admite este evento para ningún otro tipo de asignación de orígenes de eventos.

Si quiere recibir alertas cuando las asignaciones de orígenes de eventos para los servicios compatibles pasen al estado Disabled, configure una alarma en Amazon CloudWatch mediante el



evento de CloudTrail `LambdaESMDisabled`. Para obtener más información sobre la configuración de una alarma de CloudWatch, consulte [Creating CloudWatch alarms for CloudTrail events: examples](#).

La entidad `serviceEventDetails` del mensaje del evento `LambdaESMDisabled` contiene uno de los siguientes códigos de error.

### **RESOURCE\_NOT\_FOUND**

El recurso especificado en la solicitud no existe.

### **FUNCTION\_NOT\_FOUND**

La función adjunta al origen del evento no existe.

### **REGION\_NAME\_NOT\_VALID**

Un nombre de región proporcionado en la función o el origen del evento no es válido.

### **AUTHORIZATION\_ERROR**

Los permisos no se han establecido o están mal configurados.

### **FUNCTION\_IN\_FAILED\_STATE**

El código de función no se compila, se ha detectado una excepción irrecuperable o ha ocurrido una implementación incorrecta.

## Ejemplos de eventos de Lambda

Un evento representa una única solicitud de cualquier origen e incluye información sobre la operación de la API solicitada, la fecha y la hora de la operación, los parámetros de la solicitud, entre otras cosas. Los archivos de registro de CloudTrail no rastrean el orden en la pila de las llamadas a la API públicas, por lo que los eventos no aparecen en un orden específico.

El siguiente ejemplo muestra entradas de registro de CloudTrail que demuestra las acciones `GetFunction` y `DeleteFunction`.

#### Note

El `eventName` puede incluir información de la fecha y la versión, como `"GetFunction20150331"`, pero sigue haciendo referencia a la misma API pública.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:03:36Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "GetFunction",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httpplib2/0.8 (gzip)",
      "errorCode": "AccessDenied",
      "errorMessage": "User: arn:aws:iam::111122223333:user/myUserName is not
authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:111122223333:function:other-acct-function",
      "requestParameters": null,
      "responseElements": null,
      "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
      "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:04:42Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "DeleteFunction20150331",
      "awsRegion": "us-east-1",
```

```
"sourceIPAddress": "127.0.0.1",
"userAgent": "Python-httpplib2/0.8 (gzip)",
"requestParameters": {
  "functionName": "basic-node-task"
},
"responseElements": null,
"requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
"eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}
]
}
```

Para obtener información sobre el contenido de los registros de CloudTrail, consulte [Contenido de los registros de CloudTrail](#) en la Guía del usuario de AWS CloudTrail.

# Visualice las invocaciones de la función de Lambda mediante AWS X-Ray

Puede utilizar AWS X-Ray para visualizar los componentes de la aplicación, identificar cuellos de botella de rendimiento y solucionar problemas de solicitudes que dieron lugar a un error. Sus funciones de Lambda envían datos de rastreo a X-Ray, y X-Ray procesa los datos para generar un mapa de servicio y resúmenes de rastreo en los que se puede buscar.

Si ha habilitado el rastreo de X en un servicio que invoca su función, Lambda envía rastreos a X-Ray automáticamente. El servicio ascendente, como Amazon API Gateway, o una aplicación alojada en Amazon EC2 que está instrumentada con el X-Ray SDK, muestra las solicitudes entrantes y agrega un encabezado de rastreo que le indica a Lambda que envíe rastreos o no. Las trazas de los productores de mensajes anteriores, como Amazon SQS, se vinculan automáticamente a las trazas de las funciones de Lambda posteriores, lo que crea una vista integral de toda la aplicación. Para obtener más información, consulte [Seguimiento de aplicaciones basadas en eventos](#) en la Guía para desarrolladores de AWS X-Ray.

## Note

En la actualidad, el seguimiento de X-Ray no es compatible con las funciones de Lambda con Amazon Managed Streaming para Apache Kafka (Amazon MSK), Apache Kafka autoadministrado, Amazon MQ con ActiveMQ y RabbitMQ, o las asignaciones de orígenes de eventos de Amazon DocumentDB.

Para activar el seguimiento activo de la función Lambda mediante la consola, siga estos pasos:

### Cómo activar el seguimiento activo

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija Configuration (Configuración), y luego Monitoring and operations tools (Herramientas de supervisión y operaciones).
4. Elija Editar.
5. En X-Ray, active Rastreo activo.
6. Seleccione Guardar.

La función necesita permiso para cargar datos de rastreo en X-Ray. Cuando activa el rastreo activo en la consola de Lambda, Lambda agrega los permisos necesarios al [rol de ejecución](#) de la función. De lo contrario, agregue la política [AWSXRayDaemonWriteAccess](#) al rol de ejecución.

X-Ray no sigue todas las solicitudes realizadas a la aplicación. X-Ray aplica un algoritmo de muestreo para garantizar que el seguimiento sea eficiente, a la vez que proporciona una muestra representativa de todas las solicitudes. La tasa de muestreo es 1 solicitud por segundo y un 5 por ciento de las solicitudes adicionales. La frecuencia de muestreo de X-Ray no se puede configurar para las funciones.

## Comprensión de los rastros

En X-Ray, un seguimiento registra información sobre una solicitud procesada por uno o varios servicios. Lambda registra 2 segmentos por seguimiento, lo que crea dos nodos en el gráfico de servicios. La siguiente imagen resalta estos dos nodos:



El primer nodo, situado a la izquierda, representa el servicio de Lambda, que recibe la solicitud de invocación. El segundo nodo representa la función Lambda específica.

El segmento registrado para el servicio de Lambda, `AWS::Lambda`, abarca todos los pasos necesarios para preparar el entorno de ejecución de Lambda. Esto incluye programar la MicroVM, crear o desbloquear un entorno de ejecución con los recursos que configuró, así como también descargar el código de función y todas las capas.

El segmento `AWS::Lambda::Function` corresponde al trabajo realizado por la función.

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de

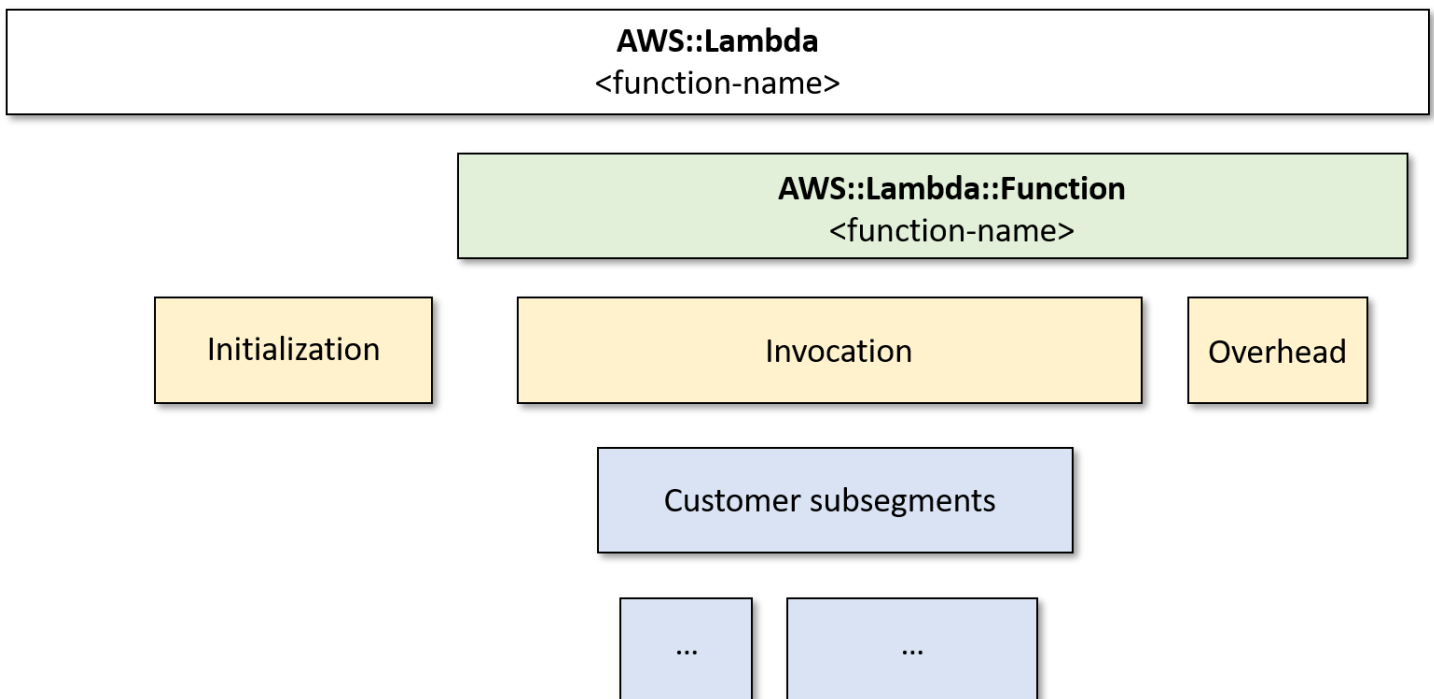
registro del sistema y los segmentos de rastreo emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

Este cambio afecta a los subsegmentos del segmento de la función. En los párrafos siguientes se describen los formatos antiguos y nuevos de estos subsegmentos.

Estos cambios se implementarán en las próximas semanas, y todas las funciones en todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastreo.

## Estructura de segmentos Lambda AWS X-Ray de estilo antiguo

La estructura de X-Ray de estilo antiguo del segmento AWS::Lambda tiene el siguiente aspecto:



En este formato, el segmento de la función tiene subsegmentos para `Initialization`, `Invocation` y `Overhead`. Solo para [Lambda SnapStart](#), también hay un subsegmento `Restore` (que no se muestra en este diagrama).

El subsegmento `Initialization` representa la fase inicial del ciclo de vida del entorno de ejecución de Lambda. Durante esta fase, Lambda inicializa las extensiones y el tiempo de ejecución y ejecuta el código de inicialización de la función.

El subsegmento `Invocation` representa la fase de invocación donde Lambda invoca el controlador de funciones. Esto comienza con el registro en tiempo de ejecución y extensión y termina cuando el motor de ejecución está listo para enviar la respuesta.

(Solo para SnapStart de Lambda) El subsegmento `Restore` muestra el tiempo que tarda Lambda en restaurar una instantánea, cargar el tiempo de ejecución (JVM) y ejecutar cualquier enlace de tiempo de ejecución de `afterRestore`. El proceso de restauración de instantáneas puede incluir el tiempo dedicado a actividades fuera de la micro VM. Esta vez se informa en el subsegmento `Restore`. No se le cobrará por el tiempo que pase fuera de la micro VM para restaurar una instantánea.

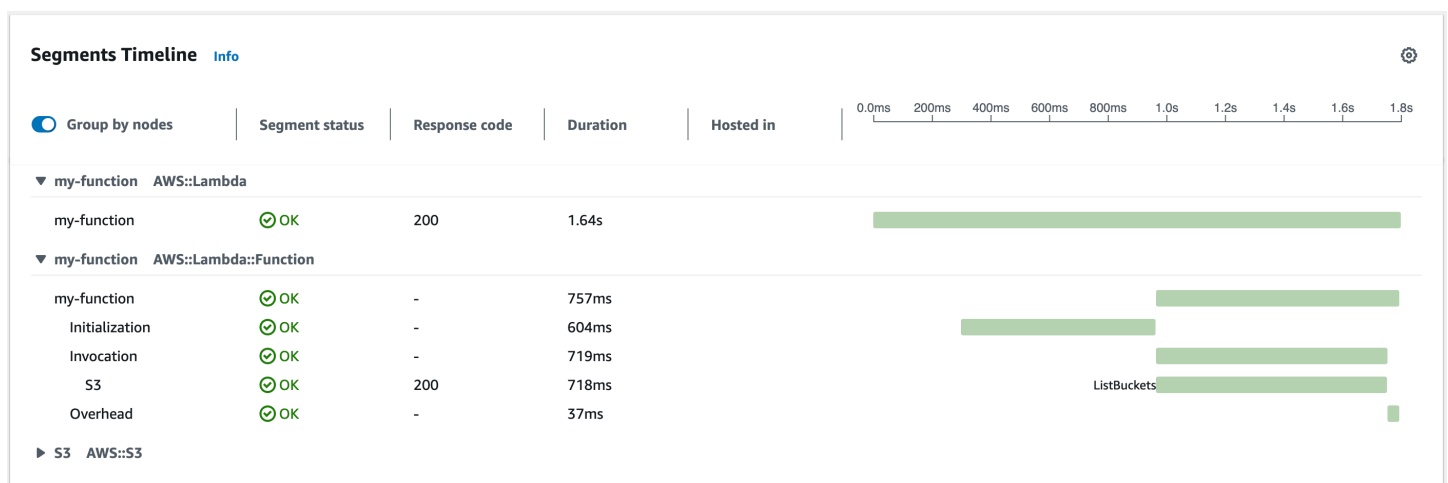
El subsegmento `Overhead` representa la fase que ocurre entre el momento en que el motor de ejecución envía la respuesta y la señal para la siguiente invocación. Durante este tiempo, el motor de ejecución finaliza todas las tareas relacionadas con una invocación y se prepara para congelar el entorno limitado.

### ⚠ Important

Puede usar el SDK de X-Ray para extender el subsegmento `Invocation` con subsegmentos adicionales para llamadas descendentes, anotaciones y metadatos. No se puede acceder al segmento de función directamente o grabar trabajo hecho fuera del ámbito de invocación del controlador.

Para obtener más información acerca de las fases del entorno de ejecución de Lambda, consulte [Entorno de ejecución](#).

En el siguiente diagrama se muestra un ejemplo de seguimiento que utiliza la estructura de X-Ray de estilo antiguo.



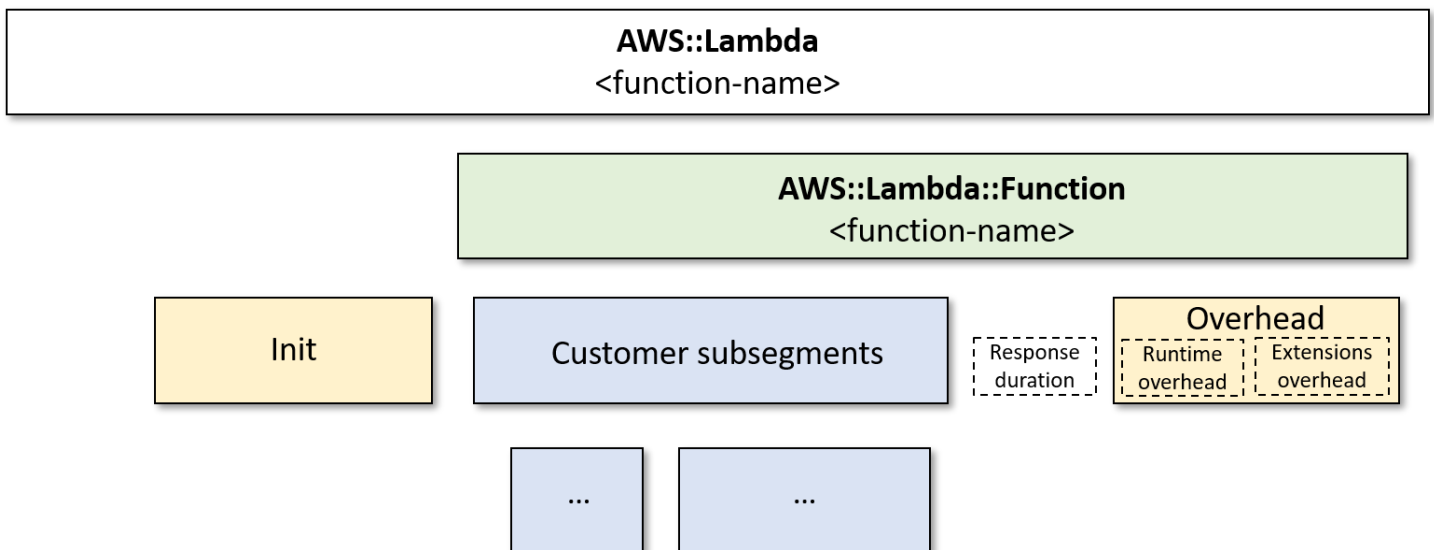
Observe los dos segmentos en el ejemplo. Ambos se denominan `my-function`, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.

### Note

Ocasionalmente, es posible que observe una gran brecha entre las fases de inicialización e invocación de la función en sus trazas de X-Ray. En el caso de las funciones que utilizan la [simultaneidad aprovisionada](#), esto se debe a que Lambda inicializa las instancias de la función mucho antes de la invocación. Para las funciones que utilizan la [simultaneidad sin reservas \(bajo demanda\)](#), Lambda puede inicializar proactivamente una instancia de función, incluso si no hay ninguna invocación. Visualmente, ambos casos se muestran como un intervalo de tiempo entre las fases de inicialización e invocación.

Estructura de segmentos Lambda AWS X-Ray de estilo nuevo

La estructura de X-Ray de estilo nuevo del segmento `AWS::Lambda` tiene el siguiente aspecto:



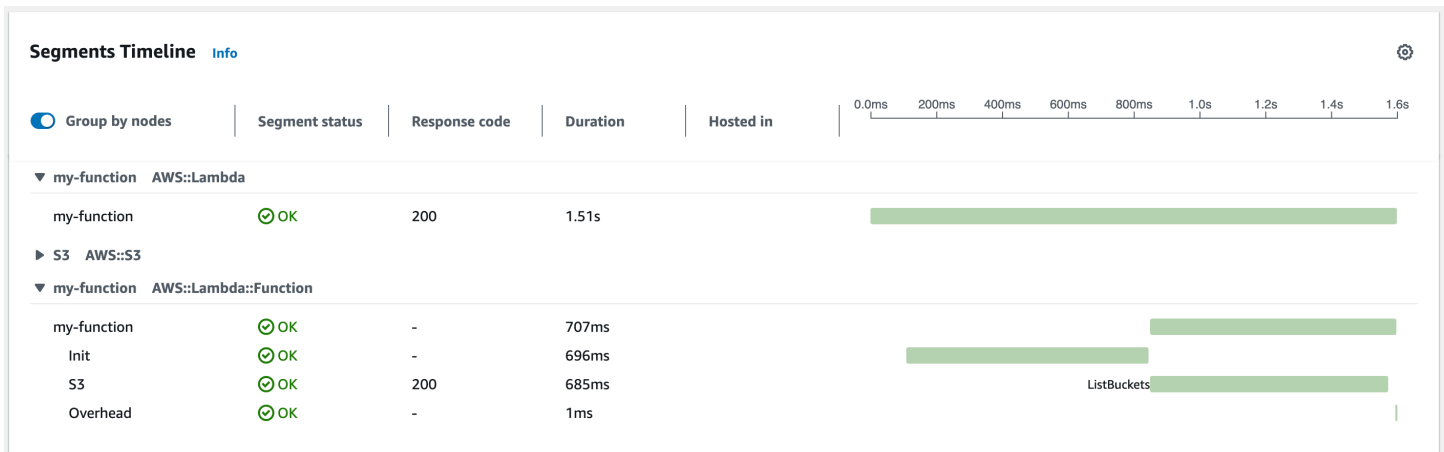
En este formato nuevo, el subsegmento `Init` representa la fase inicial del ciclo de vida del entorno de ejecución de Lambda como antes.

No hay ningún segmento de invocación en el nuevo formato. En cambio, los subsegmentos de clientes se adjuntan directamente al segmento `AWS::Lambda::Function`. Este segmento contiene las siguientes métricas como anotaciones:



- `aws.responseLatency`: el tiempo que tarda la función en ejecutarse
- `aws.responseDuration`: el tiempo necesario para transferir la respuesta al cliente
- `aws.runtimeOverhead`: la cantidad de tiempo adicional que necesitó el tiempo de ejecución para finalizar
- `aws.extensionOverhead`: la cantidad de tiempo adicional que necesitaron las extensiones para finalizar

En el siguiente diagrama se muestra un ejemplo de seguimiento que utiliza la estructura de X-Ray de estilo nuevo.



Observe los dos segmentos en el ejemplo. Ambos se denominan `my-function`, pero uno tiene un origen de `AWS::Lambda` y el otro tiene origen de `AWS::Lambda::Function`. Si el segmento `AWS::Lambda` muestra un error, el servicio Lambda tuvo un problema. Si el segmento `AWS::Lambda::Function` muestra un error, la función tuvo un problema.

Consulte los siguientes temas para obtener una introducción específica del seguimiento en Lambda:

- [Instrumentación del código Node.js en AWS Lambda](#)
- [Instrumentación del código Python en AWS Lambda](#)
- [Instrumentación del código Ruby en AWS Lambda](#)
- [Instrumentación del código Java en AWS Lambda](#)
- [Instrumentación del código Go en AWS Lambda](#)
- [Instrumentación de código C# en AWS Lambda](#)

Para obtener una lista completa de los servicios que admiten instrumentación activa, consulte [Servicios de AWS admitidos](#) en la Guía para desarrolladores de AWS X-Ray.

## Permisos de rol de ejecución

Lambda necesita los siguientes permisos para enviar datos de seguimiento a X-Ray. Añada dichos permisos al [rol de ejecución](#) de su función.

- [xray:PutTraceSegments](#)
- [xray:PutTelemetryRecords](#)

Estos permisos se incluyen en la política administrada [AWSXRayDaemonWriteAccess](#).

## El daemon de AWS X-Ray

En lugar de enviar datos de seguimiento directamente a la X-Ray API, el X-Ray SDK utiliza un proceso de daemon. El daemon de AWS X-Ray es una aplicación que se ejecuta en el entorno de Lambda y escucha el tráfico UDP que contiene segmentos y subsegmentos. Almacena en búfer los datos entrantes y los escribe en X-Ray en lotes, lo que reduce la sobrecarga de procesamiento y memoria necesaria para rastrear las invocaciones.

El tiempo de ejecución de Lambda permite al daemon hasta un 3 % de la memoria configurada de su función o 16 MB, lo que sea mayor. Si su función se queda sin memoria durante la invocación, el tiempo de ejecución termina el proceso del daemon primero para liberar memoria.

El proceso del daemon es totalmente administrado por Lambda y el usuario no puede configurarlo. Todos los segmentos generados por las invocaciones de funciones se registran en la misma cuenta que la función de Lambda. El daemon no se puede configurar para redirigirlo a ninguna otra cuenta.

Para obtener más información, consulte [El daemon de X-Ray](#) en la Guía para desarrolladores de X-Ray.

## Habilitación del seguimiento activo con la API de Lambda

Para administrar la configuración de seguimiento con la AWS CLI o el AWS SDK, utilice las siguientes operaciones de API:

- [UpdateFunctionConfiguration](#)
- [GetFunctionConfiguration](#)

- [CreateFunction](#)

El siguiente comando de ejemplo de la AWS CLI habilita el seguimiento activo en una función llamada my-function.

```
aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

El modo de seguimiento forma parte de la configuración específica de la versión, cuando se publica una versión de la función. No se puede cambiar el modo de seguimiento de una versión publicada.

## Habilitación del seguimiento activo con AWS CloudFormation

Para activar el seguimiento en un recurso de `AWS::Lambda::Function` de una plantilla de AWS CloudFormation, utilice la propiedad `TracingConfig`.

Example [función-inline.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para un recurso `AWS::Serverless::Function` de AWS Serverless Application Model (AWS SAM), utilice la propiedad `Tracing`.

Example [template.yml](#): configuración de rastreo

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

# Supervise el rendimiento de las funciones con Amazon CloudWatch Lambda Insights

Amazon CloudWatch Lambda Insights recopila y agrega métricas y registros de rendimiento en tiempo de ejecución de funciones de Lambda para las aplicaciones sin servidor. En esta página se describe cómo habilitar y utilizar Lambda Insights para diagnosticar problemas con las funciones de Lambda.

## Secciones

- [Cómo monitorea las aplicaciones sin servidor de Lambda Insights](#)
- [Precios](#)
- [Tiempos de ejecución admitidos](#)
- [Activación de Lambda Insights en la consola de Lambda](#)
- [Habilitación de Lambda Insights mediante programación](#)
- [Uso del panel de información de Lambda Insights](#)
- [Ejemplo de flujo de trabajo para detectar anomalías de función](#)
- [Ejemplo de flujo de trabajo mediante consultas para solucionar problemas de una función](#)
- [Sigüientes pasos](#)

## Cómo monitorea las aplicaciones sin servidor de Lambda Insights

CloudWatch Lambda Insights es una solución de monitoreo y solución de problemas para aplicaciones sin servidor que se ejecutan en AWS Lambda. La solución recopila, agrega y resume métricas de nivel de sistema, incluido el tiempo de CPU, la memoria, el disco y el uso de red. También recopila, agrega y resume información de diagnóstico como inicios en frío y paradas de trabajo de Lambda para ayudarle a aislar problemas con las funciones de Lambda y resolverlos rápidamente.

Lambda Insights utiliza un nuevo CloudWatch Lambda Insights [extensión](#), que se proporciona como una [capa de Lambda](#). Cuando habilita esta extensión en una función de Lambda para un tiempo de ejecución admitido, recopila métricas de nivel de sistema y emite un único evento de registro de rendimiento para cada invocación de esa función de Lambda. CloudWatch utiliza el formato de métrica incrustado para extraer métricas de los eventos de registro. Para obtener más información, consulte [Uso de extensiones de AWS Lambda](#).

La capa de Lambda Insights amplía `CreateLogStream` y `PutLogEvents` para el grupo de registros de `/aws/lambda-insights/`.

## Precios

Cuando habilita Lambda Insights para la función de Lambda, Lambda Insights notifica 8 métricas por función y cada invocación de función envía alrededor de 1 KB de datos de registro a CloudWatch. Sólo hay que pagar por las métricas y registros que Lambda Insights notifica para la función. No se requieren cuotas mínimas ni políticas obligatorias de uso del servicio.¿. No hay que pagar por Lambda Insights si la función no se invoca. Para obtener más información sobre precios, consulte [Precios de Amazon CloudWatch](#).

## Tiempos de ejecución admitidos

Puede utilizar Lambda Insights con cualquiera de los tiempos de ejecución que soportan [extensiones de Lambda](#).

## Activación de Lambda Insights en la consola de Lambda

Puede habilitar el monitoreo mejorado de Lambda Insights en funciones de Lambda nuevas y existentes. Cuando habilita Lambda Insights en una función de la consola de para un tiempo de ejecución compatible, Lambda agrega la [extensión](#) de Lambda Insights como una capa a la función y verifica o intenta adjuntar la política [CloudWatchLambdaInsightsExecutionRolePolicy](#) al [rol de ejecución](#) de la función.

Activar Lambda Insights en la consola de Lambda

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija su función.
3. Elija la pestaña Configuración.
4. En el menú de la izquierda, elija Herramientas de monitoreo y operación.
5. En el panel Herramientas adicionales de monitoreo, elija Editar.
6. En CloudWatch Lambda Insights, active Monitorización mejorada.
7. Seleccione Guardar.

## Habilitación de Lambda Insights mediante programación

También puede habilitar Lambda Insights mediante la AWS Command Line Interface (AWS CLI), AWS Serverless Application Model (SAM) CLI, AWS CloudFormation o AWS Cloud Development Kit (AWS CDK). Cuando habilita Lambda Insights mediante programación en una función para un tiempo de ejecución compatible, CloudWatch adjunta la política [CloudWatchLambdaInsightsExecutionRolePolicy](#) al [rol de ejecución](#) de la función.

Para obtener más información, consulte [Introducción a Lambda Insights](#) en la Guía del usuario de Amazon CloudWatch.

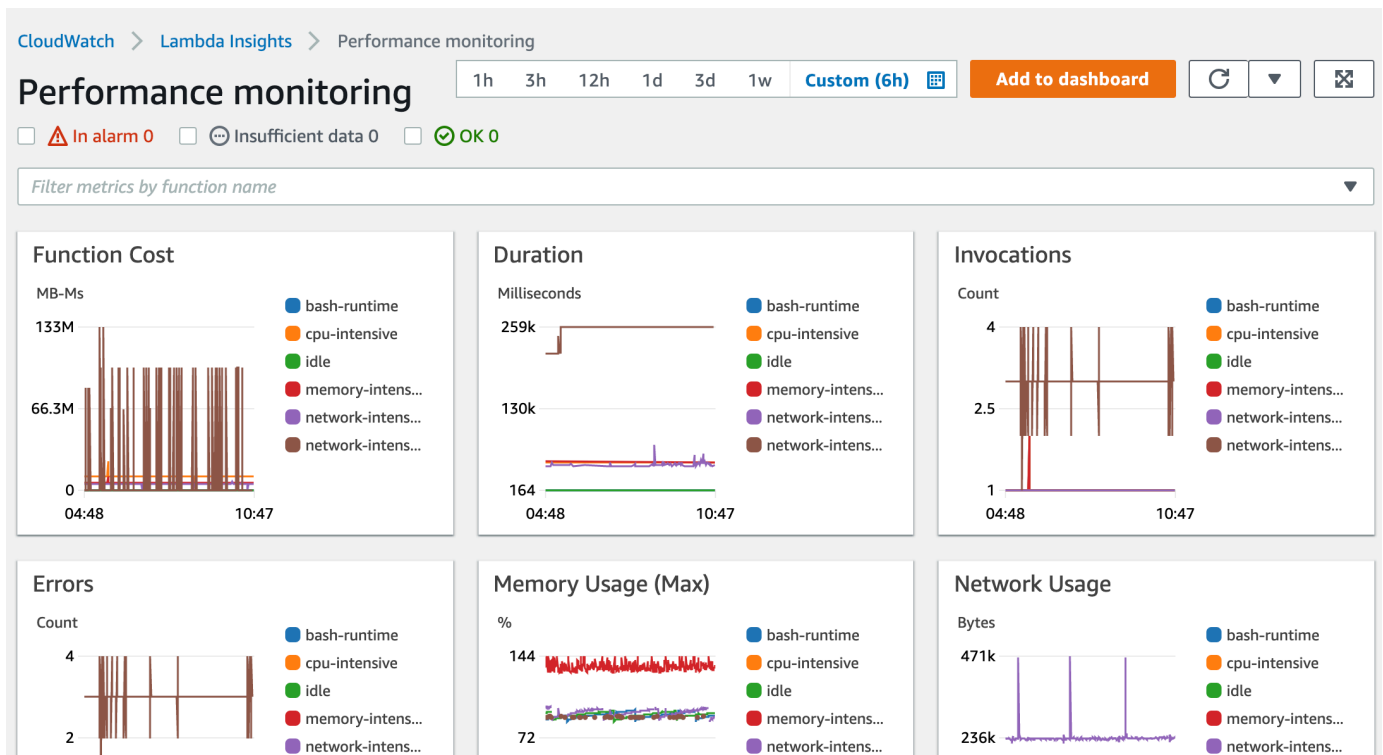
## Uso del panel de información de Lambda Insights

El panel de Lambda Insights tiene dos vistas en la consola de CloudWatch: la vista general de múltiples funciones y la vista de una sola función. La vista general de múltiples funciones agrega las métricas de tiempo de ejecución para las funciones de Lambda en la cuenta y la región de AWS actuales. La vista de una sola función muestra las métricas de tiempo de ejecución disponibles para una sola función Lambda.

Puede utilizar información general del panel multifunción de Lambda Insights de la consola de CloudWatch para identificar funciones de Lambda utilizadas en exceso o infrautilizadas. Puede utilizar la vista de una sola función del panel de Lambda Insights de la consola de CloudWatch para solucionar problemas de solicitudes individuales.

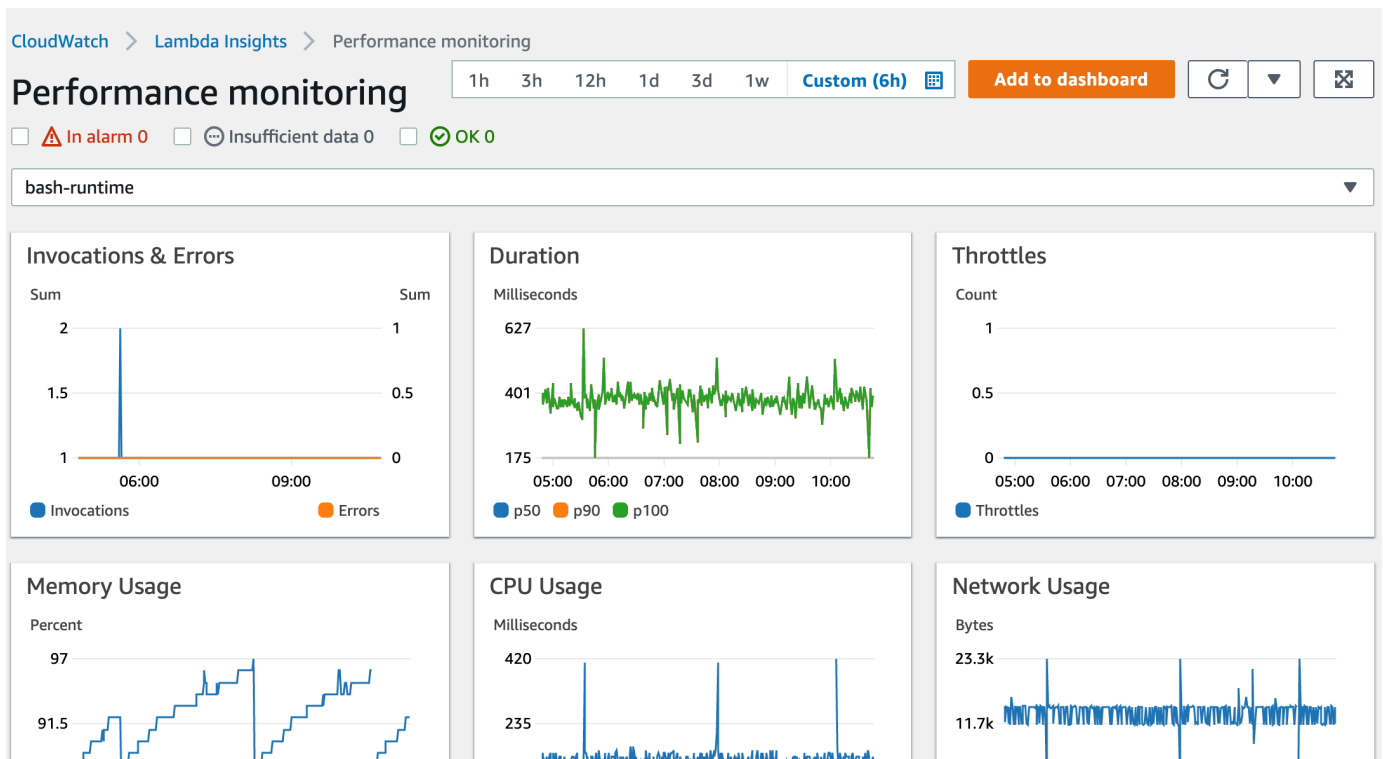
Para consultar las métricas de tiempo de ejecución de todas las funciones

1. Abra la página [Multi-function \(Múltiples funciones\)](#) de la consola de CloudWatch.
2. Elija entre los intervalos de tiempo predefinidos o elija un intervalo de tiempo personalizado.
3. (Opcional) Elija Add to dashboard (Agregar al panel) para agregar los widgets al panel de CloudWatch.



Para consultar las métricas de tiempo de ejecución de una sola función

1. Abra la página [Single-function \(Una sola función\)](#) de la consola de CloudWatch.
2. Elija entre los intervalos de tiempo predefinidos o elija un intervalo de tiempo personalizado.
3. (Opcional) Elija Add to dashboard (Agregar al panel) para agregar los widgets al panel de CloudWatch.



Para obtener más información, consulte [Creación y uso de widgets en paneles de CloudWatch](#).

## Ejemplo de flujo de trabajo para detectar anomalías de función


Puede utilizar la vista general de múltiples funciones del panel de Lambda Insights para identificar y detectar anomalías de memoria informática con la función. Por ejemplo, si la vista general de múltiples funciones indica que una función utiliza una cantidad de memoria grande, puede consultar métricas detalladas de utilización de memoria en el panel Memory Usage (Uso de memoria). A continuación, puede ir al panel de métricas para habilitar la detección de anomalías o crear una alarma.


Para habilitar la detección de anomalías para una función

1. Abra la página [Multi-function \(Múltiples funciones\)](#) de la consola de CloudWatch.
2. En Function summary (Resumen de funciones), elija el nombre de la función.

La vista de una sola función se abre con las métricas de tiempo de ejecución de la función.

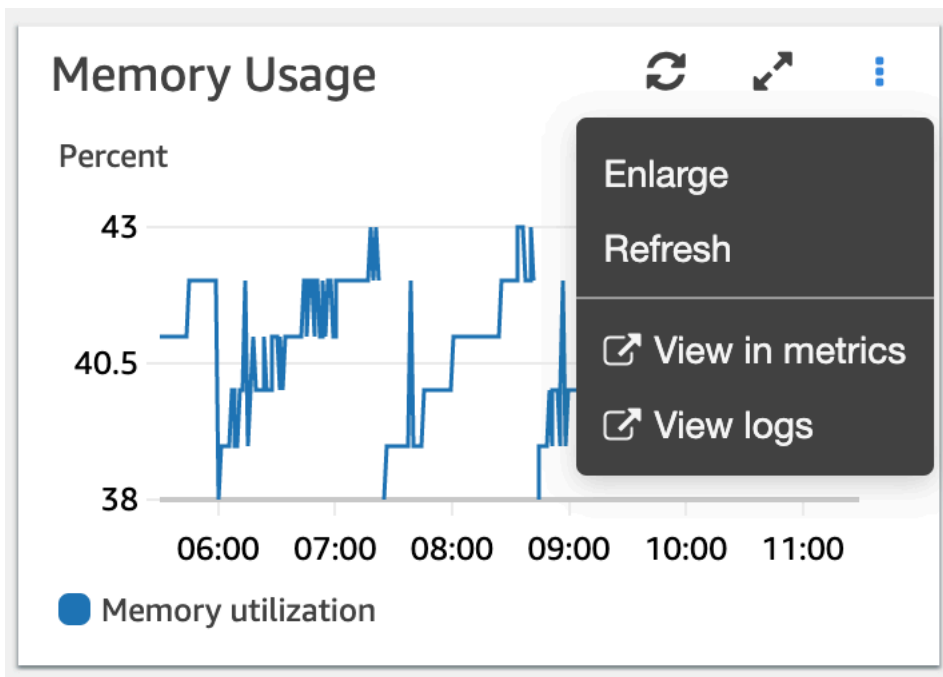


**Function summary (6)** Actions  ▼

< 1 > 

<input type="checkbox"/>	Function name ▲	Invocations ▼	CPU time ▼	Network IO ▼	Max. memory ▼	Cold starts ▼
<input type="checkbox"/>	<a href="#">bash-runtime</a>	360	132.9167ms	4770 kB	<div style="width: 97%;"><div style="width: 97%;"></div></div> 97%	3
<input type="checkbox"/>	<a href="#">cpu-intensive</a>	359	6714.2897ms	4780 kB	<div style="width: 43%;"><div style="width: 43%;"></div></div> 43%	4
<input type="checkbox"/>	<a href="#">idle</a>	359	120.2507ms	4746 kB	<div style="width: 96%;"><div style="width: 96%;"></div></div> 96%	3
<input type="checkbox"/>	<a href="#">memory-intensive</a>	358	2385.9497ms	4794 kB	<div style="width: 44%;"><div style="width: 44%;"></div></div> 44%	4
<input type="checkbox"/>	<a href="#">network-intensive</a>	359	781.0585ms	82008 kB	<div style="width: 99%;"><div style="width: 99%;"></div></div> 99%	3
<input type="checkbox"/>	<a href="#">network-intensive-vpc</a>	43	2730.6977ms	95 kB	<div style="width: 91%;"><div style="width: 91%;"></div></div> 91%	43

- En el panel Memory Usage (Uso de memoria), elija los tres puntos verticales y, a continuación, elija View in metrics (Ver en métricas) para abrir el panel Metrics (Métricas).



- En la pestaña Graphed metrics (Métricas gráficas), en la columna Actions (Acciones), elija el primer icono para habilitar la detección de anomalías para la función.

All metrics		Graphed metrics (6)		Graph options		Source				
Math expression		Dynamic labels		Statistic: Maximum		Period: 1 Minute		Remove all		
✓		Label	Details	Statistic	Period	Y Axis	Actions			
✓	■	bash-runtime	LambdaInsights * memory_utilization * functio...	Maximum	1 Minute	< >	📉	🔔	📄	✖
✓	■	cpu-intensive	LambdaInsights * memory_utilization * functio...	Maximum	1 Minute	< >	📉	🔔	📄	✖
✓	■	idle	LambdaInsights * memory_utilization * functio...	Maximum	1 Minute	< >	📉	🔔	📄	✖
✓	■	memory-intensive	LambdaInsights * memory_utilization * functio...	Maximum	1 Minute	< >	📉	🔔	📄	✖

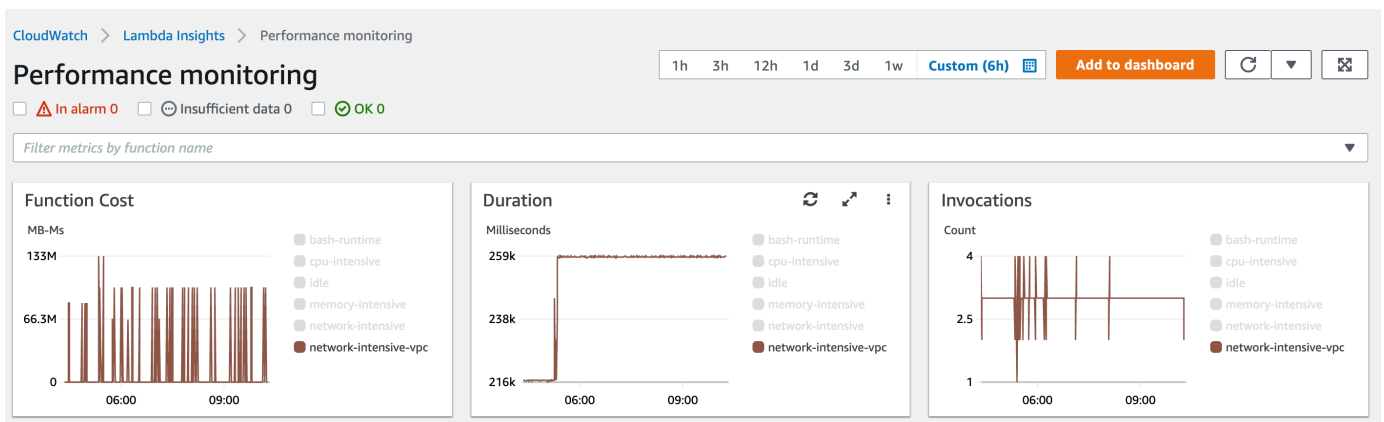
Para obtener más información, consulta [Uso de la detección de anomalías de CloudWatch](#).

## Ejemplo de flujo de trabajo mediante consultas para solucionar problemas de una función

Puede utilizar la vista de una sola función en el panel de Lambda Insights para identificar la causa raíz de un pico en la duración de la función. Por ejemplo, si la vista general de múltiples funciones indica un aumento grande en la duración de la función, puede pausar o elegir cada función en el panel de Duration (Duración) para determinar qué función está causando el aumento. A continuación, puede ir a la vista de una sola función y revisar los registros de aplicación para determinar la causa raíz.

Para ejecutar consultas en una función

1. Abra la página [Multi-function \(Múltiples funciones\)](#) de la consola de CloudWatch.
2. En el panel Duration (Duración), elija la función para filtrar las métricas de duración.



3. Abra la página [Una sola función](#).

4. Elija el menú desplegable Filter metrics by function name (Filtrar métricas por nombre de función) y, a continuación, elija la función.
5. Para consultar los Most recent 1000 application logs (1000 registros de aplicaciones más recientes), elija la pestaña Application logs (Registros de aplicaciones).
6. Revise la Timestamp (Marca temporal) y el Message (Mensaje) para identificar la solicitud de invocación que desea solucionar problemas.

Timestamp	Message
2020-09-30T16:24:36.121-06	00000000 --:--: 0:03:06 --:--: 0
2020-09-30T16:24:34.917-06	00000000 --:--: 0:04:15 --:--: 0
2020-09-30T16:24:34.120-06	00000000 --:--: 0:03:04 --:--: 0
2020-09-30T16:24:33.033-06	00000000 --:--: 0:01:26 --:--: 0

7. Para mostrar las Most recent 1000 invocations (1000 invocaciones más recientes), elija la pestaña Invocations (Invocaciones).
8. Seleccione la Timestamp (Marca temporal) o el Message (Mensaje) para la solicitud de invocación que desea solucionar problemas.

	Timestamp	Request ID	Trace	Memory %	Network IO	CPU time	Cold start
<input checked="" type="checkbox"/>	2020-09-30 16:22:34 (UTC-06:00)	247e6369-3a2b-...	-	91%	2 kB	2550ms	Yes
<input type="checkbox"/>	2020-09-30 16:13:39 (UTC-06:00)	311fb438-fa9d-4...	-	90%	2 kB	2340ms	Yes

9. Elija el menú desplegable View logs (Ver registros) y, a continuación, elija View performance logs (Ver registros de rendimiento).

Se abre una consulta autogenerada para la función en el panel de Logs Insights.

10. Elija Run query (Ejecutar consulta) para generar un mensaje de Logs (Registros) para la solicitud de invocación.

The screenshot shows the AWS Lambda Insights console interface. At the top, there is a search bar with the text "Select log group(s)" and a dropdown arrow. To the right, there are two time range selectors: "2020-09-30 (10:35:41)" and "2020-09-30 (16:35:41)". Below this, there is a text input field containing the query: "/aws/lambda-insights X" and a "Clear" button. The query itself is displayed in a code editor with line numbers 1 through 4:

```

1 fields @timestamp, @message
2 | .filter function_name = "network-intensive-vpc"
3 | .filter request_id = "247e6369-3a2b-4ccf-9e95-fb80c6ba711f"
4 | sort @timestamp desc

```

Below the query editor are three buttons: "Run query" (in orange), "Save", and "History".

The lower section of the screenshot shows the results of the query. It has two tabs: "Logs" (selected) and "Visualization". On the right, there are buttons for "Export results" and "Add to dashboard", along with a settings gear icon. The main area displays a histogram with the text "Showing 1 of 1 records matched" and "1,856 records (2.0 MB) scanned in 4.0s @ 467 records/s (521.7 kB/s)". The x-axis of the histogram shows time from 11 AM to 04 PM. Below the histogram is a table with the following columns: "#", "@timestamp", and "@message". The table contains one row:

#	@timestamp	@message
▶ 1	2020-09-30T16:22:34....	{"cpu_system_time":1520,"shutdown":1,"cpu_user_time":1030,"agent_memory_avg":7487349,"used_memory...

## Siguientes pasos

- Obtenga información sobre cómo crear un panel de CloudWatch Logs en [Create a Dashboard \(Crear un panel\)](#) en la Guía del usuario de Amazon CloudWatch.
- Obtenga información sobre cómo agregar consultas a un panel de CloudWatch Logs en [Add Query to Dashboard or Export Query Results \(Agregar consulta al panel o Exportar resultados de consulta\)](#) en la Guía del usuario de Amazon CloudWatch.

# Administración de las dependencias de Lambda con capas

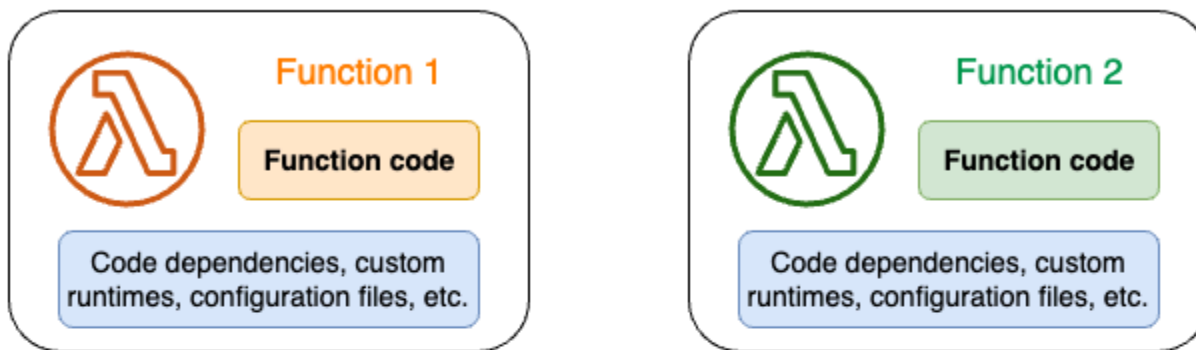
Una capa de Lambda es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración.

Hay varias razones por las que podría considerar la posibilidad de usar las capas:

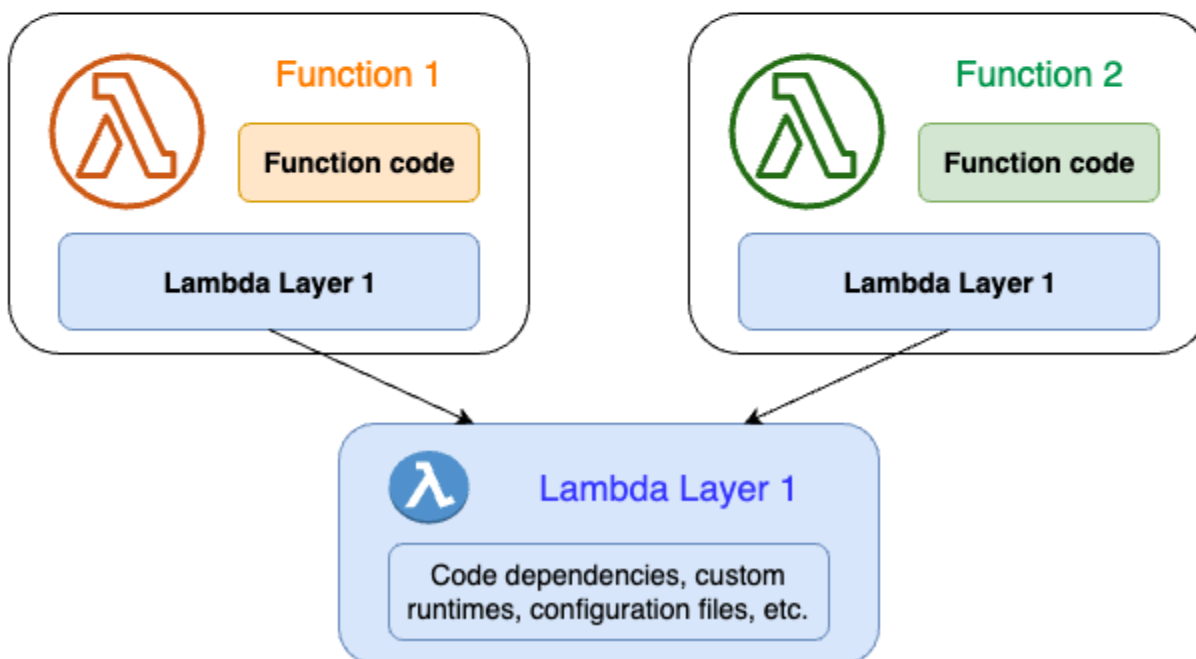
- Para reducir el tamaño de sus paquetes de implementación. En lugar de incluir todas las dependencias de la función junto con el código de la función en el paquete de implementación, colóquelas en una capa. Esto mantiene los paquetes de implementación pequeños y organizados.
- Para separar la lógica de las funciones principales de las dependencias. Con las capas, puede actualizar las dependencias de las funciones independientemente del código de la función y viceversa. Esto promueve la separación de preocupaciones y lo ayuda a concentrarse en la lógica de su función.
- Para compartir dependencias entre varias funciones. Después de crear una capa, puede aplicarla a cualquier número de funciones de su cuenta. Sin capas, debe incluir las mismas dependencias en cada paquete de implementación individual.
- Para usar el editor de código de la consola de Lambda. El editor de código es una herramienta útil para probar rápidamente actualizaciones de código de funciones menores. Sin embargo, no puede usar el editor si el tamaño del paquete de implementación es demasiado grande. El uso de capas reduce el tamaño del paquete y puede desbloquear el uso del editor de código.

El siguiente diagrama ilustra las diferencias arquitectónicas de alto nivel entre dos funciones que comparten dependencias. Una usa capas Lambda y la otra no.

## Lambda function components: Without layers



## Lambda function components: With layers



Cuando incluye una capa en una función, Lambda extrae los contenidos de la capa en el directorio `/opt` en el [entorno de ejecución](#) de la función. Todos los tiempos de ejecución de Lambda compatibles de forma nativa incluyen rutas a directorios específicos dentro del directorio `/opt`. Esto permite que la función acceda al contenido de la capa. Para obtener más información sobre estas rutas específicas y sobre cómo empaquetar correctamente las capas, consulte [the section called “Empaquetado de las capas”](#).

Puede incluir hasta cinco capas por función. Además, puede utilizar capas solo con funciones de Lambda [implementadas como archivos .zip](#). Para una función [definida como una imagen de](#)

[contenedor](#), empaquete su tiempo de ejecución preferido y todas las dependencias de código al crear la imagen de contenedor. Para obtener más información, consulte [Cómo trabajar con capas y extensiones de Lambda](#) en imágenes de contenedor en el blog de cómputo de AWS.

## Temas

- [Cómo usar las capas](#)
- [Capas y versiones de capas](#)
- [Empaquetado del contenido de la capa](#)
- [Creación y eliminación de capas en Lambda](#)
- [Adición de capas a las funciones](#)
- [Usar AWS CloudFormation con capas](#)
- [Usar AWS SAM con capas](#)

## Cómo usar las capas

Para crear una capa, empaquete sus dependencias en un archivo .zip, de forma similar a como [crear un paquete de implementación normal](#). Más específicamente, el proceso general de creación y uso de capas incluye estos tres pasos:

- Primero, empaquete el contenido de la capa. Esto significa crear un archivo .zip. Para obtener más información, consulte [the section called “Empaquetado de las capas”](#).
- A continuación, cree la capa en Lambda. Para obtener más información, consulte [the section called “Creación y eliminación de capas”](#).
- Agregue la capa a las funciones. Para obtener más información, consulte [the section called “Adición de capas”](#).

## Capas y versiones de capas

Una versión de capa es una instantánea inmutable de una versión específica de una capa. Al crear una capa nueva, Lambda crea una nueva versión de capa con el número de versión 1. Cada vez que publique una actualización de la capa, Lambda incrementa el número de versión y crea una nueva versión de capa.

Cada versión de capa se identifica mediante un Nombre de recurso de Amazon (ARN) único. Al agregar una capa a la función, debe especificar la versión de capa exacta que desea utilizar.

## Empaquetado del contenido de la capa

Una capa de Lambda es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración.

En esta sección se explica cómo empaquetar correctamente el contenido de la capa. Para obtener más información conceptual sobre las capas y los motivos por los que podría considerar la posibilidad de utilizarlas, consulte [Capas de Lambda](#).

El primer paso para crear una capa consiste en agrupar todo el contenido de la capa en un archivo .zip. Dado que las funciones de Lambda se ejecutan en [Amazon Linux](#), el contenido de la capa debe poder compilarse y crearse en un entorno de Linux.

Para garantizar que el contenido de la capa funcione correctamente en un entorno de Linux, se recomienda crear el contenido de la capa con una herramienta como [Docker](#) o [AWS Cloud9](#). AWS Cloud9 es un entorno de desarrollo integrado (IDE) basado en la nube que proporciona acceso integrado a un servidor Linux para ejecutar y probar código. Para obtener más información, consulte [Using Lambda layers to simplify your development process](#) en AWS Compute Blog.

### Temas

- [Rutas de capa para cada tiempo de ejecución de Lambda](#)

## Rutas de capa para cada tiempo de ejecución de Lambda

Cuando agrega una capa a una función, Lambda carga el contenido de la capa en el directorio /opt de ese entorno de ejecución. Para cada tiempo de ejecución de Lambda, la variable PATH ya incluye rutas de carpeta específicas en el directorio /opt. Para garantizar que la variable PATH recoja el contenido de la capa, el archivo .zip de la capa, debe tener sus dependencias en las siguientes rutas de carpeta:

Tiempo de ejecución	Ruta
Node.js	nodejs/node_modules
	nodejs/node16/node_modules (NODE_PATH )
	nodejs/node18/node_modules (NODE_PATH )



Tiempo de ejecución	Ruta
	nodejs/node20/node_modules (NODE_PATH )
Python	python python/lib/ <i>python3.x</i> /site-packages (directorios del sitio)
Java	java/lib (CLASSPATH )
Ruby	ruby/gems/3.2.0 (GEM_PATH) ruby/lib (RUBYLIB)
Todos los tiempos de ejecución	bin (PATH) lib (LD_LIBRARY_PATH )

En los siguientes ejemplos se muestra cómo puede estructurar las carpetas en el archivo .zip de su capa.

## Node.js

Example estructura de archivos para el SDK de AWS X-Ray para Node.js

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

## Python

Example estructura de archivos para la biblioteca de solicitudes

```
layer_content.zip
# python
# lib
# python3.11
# site-packages
# requests
# <other_dependencies> (i.e. dependencies of the requests package)
```

```
# ...
```

## Ruby

### Example estructura de archivos para la gema de JSON

```
json.zip
# ruby/gems/2.7.0/
  | build_info
  | cache
  | doc
  | extensions
  | gems
  | # json-2.1.0
# specifications
  # json-2.1.0.gemspec
```

## Java

### Example estructura de archivos para el archivo JAR de Jackson

```
layer_content.zip
# java
  # lib
    # jackson-core-2.17.0.jar
    # <other potential dependencies>
    # ...
```

## All

### Example estructura de archivos para la biblioteca jq

```
jq.zip
# bin/jq
```

Para obtener instrucciones específicas del idioma sobre cómo empaquetar, crear y agregar una capa, consulte las siguientes páginas:

- Python: [the section called “Capas”](#)
- Java: [the section called “Capas”](#)



# Creación y eliminación de capas en Lambda

Una capa de Lambda es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración.

En esta sección se explica cómo crear y eliminar capas en Lambda. Para obtener más información conceptual sobre las capas y los motivos por los que podría considerar la posibilidad de utilizarlas, consulte [Capas de Lambda](#).

Después de haber [empaquetado el contenido de su capa](#), el siguiente paso consiste en crear la capa en Lambda. En esta sección se muestra cómo crear y eliminar capas utilizando únicamente la consola de Lambda o la API de Lambda. Para crear una capa con AWS CloudFormation, consulte [the section called “Capas con AWS CloudFormation”](#). Para crear una capa con AWS Serverless Application Model (AWS SAM), consulte [the section called “Capas con AWS SAM”](#).

## Temas

- [Creación de una capa](#)
- [Eliminación de una versión de capa](#)

## Creación de una capa

Para crear una capa, puede cargar el archivo .zip desde su equipo local o desde Amazon Simple Storage Service (Amazon S3). Lambda extrae el contenido de la capa en el directorio /opt al configurar el entorno de ejecución para la función.

Las capas pueden tener una o más [versiones de capa](#). Al crear una capa, Lambda establece la versión de la capa en la versión 1. Puede cambiar los permisos de una versión de capa existente en cualquier momento. Sin embargo, para actualizar el código o realizar otros cambios de configuración, debe crear una nueva versión de la capa.

Para crear una capa (consola)

1. Abra la página de [Capas](#) de la consola de Lambda.
2. Elija Crear capa.
3. En Configuración de la capa, en Nombre, escriba un nombre para la capa.
4. (Opcional) En Descripción, escriba una descripción para su capa.

5. Para cargar el código de capa, realice una de las siguientes acciones:
  - Para cargar un archivo.zip desde el equipo, elija Cargar un archivo .zip. Seleccione Cargar para seleccionar el archivo .zip local.
  - Para cargar un archivo desde Amazon S3, elija Upload a file from Amazon S3 (Cargar un archivo desde Amazon S3). Entonces, para el URL de enlace de Amazon S3, ingrese un enlace al archivo.
6. (Opcional) En Arquitecturas compatibles, elija un valor o ambos valores. Para obtener más información, consulte [the section called “Conjuntos de instrucciones \(ARM/x86\)”](#).
7. (Opcional) Para Tiempos de ejecución compatibles, elija los tiempos de ejecución con los que la capa es compatible.
8. (Opcional) Para Licencia, introduzca la información de licencia necesaria.
9. Seleccione Crear.

Como alternativa, también puedes usar la [PublishLayerVersion](#) API para crear una capa. Por ejemplo, puede utilizar el comando `publish-layer-version` de la AWS Command Line Interface (CLI) con un nombre, una descripción y un archivo .zip especificados. La información de la licencia, los tiempos de ejecución compatibles y los parámetros de arquitectura compatibles son opcionales.

```
aws lambda publish-layer-version --layer-name my-layer \  
  --description "My layer" \  
  --license-info "MIT" \  
  --zip-file fileb://layer.zip \  
  --compatible-runtimes python3.10 python3.11 \  
  --compatible-architectures "arm64" "x86_64"
```

Debería ver una salida similar a esta:

```
{  
  "Content": {  
    "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/  
snapshots/123456789012/my-layer-4aaa2fbb-ff77-4b0a-ad92-5b78a716a96a?  
versionId=27iWyA73cCAYqyH...",  
    "CodeSha256": "tv9jJ0+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",  
    "CodeSize": 169  
  },  
  "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",  
  "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:1",  
  "Description": "My layer",
```

```
"CreateDate": "2023-11-14T23:03:52.894+0000",
"Version": 1,
"CompatibleArchitectures": [
  "arm64",
  "x86_64"
],
"LicenseInfo": "MIT",
"CompatibleRuntimes": [
  "python3.10",
  "python3.11"
]
}
```

Cada vez que llame a `publish-layer-version`, crea una nueva versión de la capa.

## Eliminación de una versión de capa

Para eliminar una versión de capa, usa la [DeleteLayerVersion](#) API. Por ejemplo, puede utilizar el comando `delete-layer-version` de la CLI con el nombre de capa y la versión de capa especificados.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Al eliminar una versión de una capa, ya no puede configurar una función de Lambda para usarla. Sin embargo, cualquier función que ya utilice la versión sigue teniendo acceso a la misma. Además, Lambda nunca reutiliza los números de versión para el nombre de una capa.

## Adición de capas a las funciones

Una capa de Lambda es un archivo .zip que contiene código o datos adicionales. Las capas suelen contener dependencias de biblioteca, un [tiempo de ejecución personalizado](#) o archivos de configuración.

En esta sección se explica cómo agregar una capa a una función de Lambda. Para obtener más información conceptual sobre las capas y los motivos por los que podría considerar la posibilidad de utilizarlas, consulte [Capas de Lambda](#).

Para poder configurar una función de Lambda en la que se utilice una capa, debe hacer lo siguiente:

- [Empaquete el contenido de su capa](#).
- [Cree una capa en Lambda](#).
- Asegúrese de que tiene permiso para llamar a la API [GetLayerVersion](#) en la versión de la capa. Para las funciones de su Cuenta de AWS, debe tener este permiso en su [política de usuario](#). Para utilizar una capa en otra cuenta, el propietario de esa cuenta debe conceder permiso a su cuenta en una [política basada en recursos](#). Para ver ejemplos, consulte [the section called “Acceso a las capas para otras cuentas”](#).

Puede agregar hasta cinco capas a una función de Lambda. El tamaño total descomprimido de la función y todas las capas no puede superar la cuota de tamaño del paquete de implementación sin comprimir, de 250 MB. Para obtener más información, consulte [Cuotas de Lambda](#).

Sus funciones pueden seguir utilizando cualquier versión de capa que ya haya agregado, incluso después de que se haya eliminado esa versión de capa o después de revocar su permiso de acceso a la capa. Sin embargo, no puede crear una nueva función que utilice una versión de capa eliminada.

### Note

Asegúrese de que las capas que agrega a una función sean compatibles con el tiempo de ejecución y la arquitectura del conjunto de instrucciones de la función.

Para agregar una capa a una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función que desea configurar.

3. En Layers (Capas), elija Add a layer (Agregar una capa)
4. En Elegir una capa, elija un origen de capa:
  - a. Para los orígenes de capa Capas de AWS o Capas personalizadas, elija una capa en el menú desplegable. En Version (Versión), elija una versión de capa en el menú desplegable.
  - b. Para el origen de capa Especificar un ARN, ingrese un ARN en el cuadro de texto y elija Verificar. A continuación, elija Agregar.

El orden en el que agrega las capas es el orden en que Lambda combina el contenido de las capas en el entorno de ejecución. Puede cambiar el orden de fusión de capas mediante la consola.

Para actualizar el orden de combinación de capas de una función (consola)

1. Abra la [página de Funciones](#) en la consola de Lambda.
2. Elija la función que desea configurar.
3. En Layers (Capas), elija Edit (Editar).
4. Elija una de las capas.
5. Elija Merge earlier (Fusionar antes) o Merge later (Fusionar después) para ajustar el orden de las capas.
6. Seleccione Guardar.

Las capas están versionadas. El contenido de cada versión de capa es inmutable. El propietario de una capa puede lanzar una nueva versión de capa para proporcionar contenido actualizado. Puede utilizar la consola para actualizar la versión de capa adjunta a sus funciones.

Para actualizar las versiones de las capas de la función (consola)

1. Abra la página de [Capas](#) de la consola de Lambda.
2. Elija la capa para la que desea actualizar la versión.
3. Elija la pestaña Funciones que utilizan esta versión.
4. Elija las funciones que desee modificar y, a continuación, elija Editar.
5. En Versión de la capa, seleccione la versión de capa a la que desea cambiar.
6. Elija Update functions (Actualizar funciones).

No se pueden actualizar las versiones de capas de función entre cuentas de AWS.



## Temas

- [Acceso al contenido de la capa desde la función](#)
- [Búsqueda de información de capa](#)

## Acceso al contenido de la capa desde la función

Si la función de Lambda incluye capas, Lambda extrae los contenidos de la capa en el directorio `/opt` en el entorno de ejecución de la función. Lambda extrae las capas en el orden (de menor a mayor) que la función indica. Lambda combina carpetas con el mismo nombre. Si el mismo archivo aparece en varias capas, la función utiliza la versión de la última capa extraída.

Cada tiempo de ejecución de Lambda agrega carpetas del directorio `/opt` específicas a la variable `PATH`. El código de la función puede acceder al contenido de la capa sin necesidad de especificar la ruta. Para obtener más información acerca de la configuración de rutas en el entorno de ejecución de Lambda, consulte [the section called “Variables definidas de entorno de tiempo de ejecución”](#).

Consulte [the section called “Rutas de capa para cada tiempo de ejecución de Lambda”](#) para saber dónde incluir las bibliotecas al crear una capa.

Si utiliza un tiempo de ejecución de Node.js o Python, puede usar el editor de código integrado en la consola de Lambda. Debería poder importar cualquier biblioteca que haya agregado como capa a la función actual.

## Búsqueda de información de capa

Para buscar capas en su cuenta que sean compatibles con el tiempo de ejecución de su función, utilice la API [ListLayers](#). Por ejemplo, puede usar el siguiente comando `list-layers` de la AWS Command Line Interface (CLI):

```
aws lambda list-layers --compatible-runtime python3.9
```

Debería ver una salida similar a esta:

```
{
  "Layers": [
    {
      "LayerName": "my-layer",
      "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",

```

```

    "LatestMatchingVersion": {
      "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
      "Version": 2,
      "Description": "My layer",
      "CreateDate": "2023-11-15T00:37:46.592+0000",
      "CompatibleRuntimes": [
        "python3.9",
        "python3.10",
        "python3.11",
      ]
    }
  ]
}

```

Para enumerar todas las capas de su cuenta, omita la opción `--compatible-runtime`. Los detalles de la respuesta muestran la versión más reciente de cada capa.

También puede obtener la versión más reciente de una capa con la API [ListLayerVersions](#). Por ejemplo, puede usar el siguiente comando `list-layer-versions` de la CLI:

```
aws lambda list-layer-versions --layer-name my-layer
```

Debería ver una salida similar a esta:

```

{
  "LayerVersions": [
    {
      "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
      "Version": 2,
      "Description": "My layer",
      "CreateDate": "2023-11-15T00:37:46.592+0000",
      "CompatibleRuntimes": [
        "java11"
      ]
    },
    {
      "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:1",
      "Version": 1,

```

```
    "Description": "My layer",
    "CreateDate": "2023-11-15T00:27:46.592+0000",
    "CompatibleRuntimes": [
      "java11"
    ]
  }
]
```

## Usar AWS CloudFormation con capas

Puede utilizar AWS CloudFormation para crear una capa y asociar la capa con la función de Lambda. En la siguiente plantilla de ejemplo se crea una capa denominada `my-lambda-layer` y se asocia la capa a la función de Lambda mediante la propiedad `Layers`.

```
---
Description: CloudFormation Template for Lambda Function with Lambda Layer
Resources:
  MyLambdaLayer:
    Type: AWS::Lambda::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      Content:
        S3Bucket: amzn-s3-demo-bucket
        S3Key: my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-lambda-function
      Runtime: python3.9
      Handler: index.handler
      Timeout: 10
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Layers:
        - !Ref MyLambdaLayer
```

## Usar AWS SAM con capas

Puede utilizar AWS Serverless Application Model (AWS SAM) para automatizar la creación de capas en su aplicación. El tipo de recurso `AWS::Serverless::LayerVersion` crea una versión de capa a la que puede hacer referencia desde la configuración de la función de Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: 'AWS::Serverless-2016-10-31'  
Description: AWS SAM Template for Lambda Function with Lambda Layer
```

### Resources:

#### MyLambdaLayer:

```
Type: AWS::Serverless::LayerVersion
```

#### Properties:

```
LayerName: my-lambda-layer
```

```
Description: My Lambda Layer
```

```
ContentUri: s3://amzn-s3-demo-bucket/my-layer.zip
```

#### CompatibleRuntimes:

```
- python3.9
```

```
- python3.10
```

```
- python3.11
```

#### MyLambdaFunction:

```
Type: AWS::Serverless::Function
```

#### Properties:

```
FunctionName: MyLambdaFunction
```

```
Runtime: python3.9
```

```
Handler: app.handler
```

```
CodeUri: s3://amzn-s3-demo-bucket/my-function
```

#### Layers:

```
- !Ref MyLambdaLayer
```

# Aumente las funciones de Lambda utilizando extensiones de Lambda

Puede usar extensiones de Lambda para aumentar las funciones Lambda. Por ejemplo, use extensiones de Lambda para integrar funciones con sus herramientas de monitoreo, observabilidad, seguridad y gobierno preferidas. Puede elegir entre un amplio conjunto de herramientas que proporciona [Partners AWS Lambda](#) o puede [crear sus propias extensiones de Lambda](#).

Lambda admite extensiones externas e internas. Una extensión externa se ejecuta como un proceso independiente en el entorno de ejecución y continúa ejecutándose después de que la invocación de la función se procese completamente. Dado que las extensiones se ejecutan como procesos separados, puede escribirlas en un idioma diferente al de la función. Todas las extensiones de soporte de [Tiempos de ejecución de Lambda](#).

Una extensión interna se ejecuta como parte del proceso de tiempo de ejecución. La función accede a extensiones internas mediante el uso de "wrapper scripts" o mecanismos en proceso tales como `JAVA_TOOL_OPTIONS`. Para obtener más información, consulte [Modificación del entorno de tiempo de ejecución](#).

Puede agregar extensiones a una función mediante la consola de Lambda AWS Command Line Interface (AWS CLI) o infraestructura como código (IaC) y herramientas como AWS CloudFormation, AWS Serverless Application Model (AWS SAM) y Terraform.

Se le cobra por el tiempo de ejecución que consume la extensión (en incrementos de 1 ms). La instalación de sus propias extensiones no supone costo alguno. Para obtener más información sobre precios de extensiones, consulte [AWS Lambda Precios](#). Para obtener información acerca de los precios de las extensiones de los socios, consulte los sitios web de dichos socios. Consulte [the section called "Socios de extensiones"](#) para obtener una lista de las extensiones oficiales de los socios.

Para ver un tutorial sobre las extensiones y cómo usarlas con las funciones de Lambda, consulte el [taller de extensiones AWS Lambda](#).

## Temas

- [Entorno de ejecución](#)
- [Impacto de desempeño y recursos](#)

- [Permisos](#)
- [Configuración de extensiones de Lambda](#)
- [Socios de extensiones de AWS Lambda](#)
- [Utilice la API de las extensiones de Lambda para crear extensiones](#)
- [Acceso a datos de telemetría en tiempo real para extensiones mediante la API de telemetría](#)

## Entorno de ejecución

Lambda invoca la función en un [entorno de ejecución](#), que proporciona un entorno en tiempo de ejecución seguro y aislado. El entorno de ejecución administra los recursos necesarios para ejecutar la función y proporciona soporte de ciclo de vida para el tiempo de ejecución y las extensiones de la función.

El ciclo de vida del entorno de ejecución incluye las siguientes fases:

- **Init**: durante esta fase, Lambda crea o descongela un entorno de ejecución con los recursos que ha configurado, descarga el código de función y todas las capas, inicializa las extensiones, inicializa el tiempo de ejecución y ejecuta el código de inicialización de la función (el código fuera del controlador principal). La fase Init ocurre durante la primera invocación, o antes de las invocaciones de función si ha habilitado [simultaneidad aprovisionada](#).

La fase Init se divide en tres subfases: `Extension init`, `Runtime init` y `Function init`. Estas subfases garantizan que todas las extensiones y el tiempo de ejecución completen sus tareas de configuración antes de que se ejecute el código de función.

Si [Lambda SnapStart](#) está activada, la fase Init ocurre cuando publica una versión de la función. Lambda guarda una instantánea del estado de la memoria y del disco del entorno de ejecución iniciado, conserva la instantánea cifrada y la almacena en caché para acceder a ella con baja latencia. Si tiene un [enlace de tiempo de ejecución](#) `beforeCheckpoint`, el código se ejecuta al final de la fase Init.

- **Restore** (solo SnapStart): cuando se invoca por primera vez una función [SnapStart](#) y, a medida que la función escala, Lambda vuelve a activar los nuevos entornos de ejecución a partir de la instantánea conservada, en lugar de activar la función desde cero. Si tiene un [enlace de tiempo de ejecución](#) `afterRestore()`, el código se ejecuta al final de la fase Restore. Se le cobrará por la duración de los enlaces de tiempo de ejecución `afterRestore()`. El tiempo de ejecución (JVM) debe cargarse, y los enlaces del tiempo de ejecución `afterRestore()` deben completarse antes

de que transcurra el tiempo de espera (10 segundos). De lo contrario, obtendrá una excepción `SnapStartTimeoutException`. Cuando se completa la fase `Restore`, Lambda invoca el controlador de funciones (la fase [Fase "invoke"](#)).

- **Invoke:** en esta fase, Lambda invoca el controlador de funciones. Después de que la función se ejecuta hasta su finalización, Lambda se prepara para manejar otra invocación de función.
- **Shutdown:** esta fase se activa si la función Lambda no recibe ninguna invocación durante un período de tiempo. En la fase `Shutdown`, Lambda apaga el tiempo de ejecución, alerta a las extensiones para que se detengan limpiamente y, a continuación, elimina el entorno. Lambda envía un evento `Shutdown` a cada extensión, lo que indica a la extensión que el entorno está a punto de cerrarse.

Durante la fase `Init`, Lambda extrae capas que contienen extensiones en el directorio `/opt` en el entorno de ejecución. Lambda busca extensiones en el directorio `/opt/extensions/`, interpreta cada archivo como un arranque ejecutable para iniciar la extensión e inicia todas las extensiones en paralelo.

## Impacto de desempeño y recursos

El tamaño de las extensiones de la función cuenta para el límite de tamaño del paquete de implementación. Para un archivo `.zip`, el tamaño total descomprimido de la función y todas las extensiones no puede superar el límite de tamaño del paquete de implementación descomprimido de 250 MB.

Las extensiones pueden afectar al rendimiento de la función porque comparten recursos de función como la CPU, la memoria y el almacenamiento. Por ejemplo, si una extensión realiza operaciones de procesamiento intensivo, es posible que la duración de ejecución de la función aumente.

Cada extensión debe completar su inicialización antes de que Lambda invoque la función. Por lo tanto, una extensión que consume un tiempo de inicialización significativo puede aumentar la latencia de la invocación de la función.

Para medir el tiempo adicional que tarda la extensión después de la ejecución de la función, puede utilizar la [métrica de la función](#) `PostRuntimeExtensionsDuration`. Para medir el aumento de la memoria utilizada, puede usar la métrica `MaxMemoryUsed`. Para conocer el impacto de una extensión específica, puede ejecutar diferentes versiones de sus funciones en paralelo.



## Permisos

Las extensiones tienen acceso a los mismos recursos que las funciones. Dado que las extensiones se ejecutan en el mismo entorno que la función, los permisos se comparten entre la función y la extensión.

Para un archivo .zip, puede crear una plantilla de AWS CloudFormation para simplificar la tarea de asociar la misma configuración de extensión, incluidos los permisos de AWS Identity and Access Management (IAM), en varias funciones.

# Configuración de extensiones de Lambda

## Configuración de extensiones (archivo de archivo .zip)

Puede agregar una extensión a su función como [capa de Lambda](#). El uso de capas permite compartir extensiones en toda la organización o con toda la comunidad de desarrolladores de Lambda. Puede agregar una o más extensiones a una capa. Puede registrar hasta 10 extensiones para una función.

Se agrega la extensión a la función utilizando el mismo método que se utilizaría para cualquier capa. Para obtener más información, consulte [Capas de Lambda](#).

Agregar una extensión a la función (consola)

1. Abra la página de [Funciones](#) en la consola de Lambda.
2. Elija una función.
3. Elija la pestaña Código si aún no está seleccionada.
4. En Capas, elija Editar.
5. En Choose a layer (Elegir una capa), elija Specify an ARN (Especificar un ARN).
6. En Specify an ARN (Especificar un ARN), escriba el nombre de recurso de Amazon (ARN) de una capa de extensión.
7. Elija Añadir.

## Uso de extensiones en imágenes de contenedor

Puede agregar extensiones a la [imagen de contenedor](#). La configuración de imagen de contenedor ENTRYPOINT especifica el proceso principal de la función. Configure el valor ENTRYPOINT en el Dockerfile o como una anulación en la configuración de la función.

Puede ejecutar varios procesos dentro de un contenedor. Lambda administra el ciclo de vida del proceso principal y cualquier otro proceso adicional. Lambda utiliza la [API de extensiones](#) para administrar el ciclo de vida de la extensión.

### Ejemplo: Agregar una extensión externa

Una extensión externa se ejecuta en un proceso independiente de la función de Lambda. Lambda inicia un proceso para cada extensión en el directorio de `/opt/extensions/`. Lambda utiliza la API

de extensiones para administrar el ciclo de vida de la extensión. Después de que la función se ha ejecutado hasta su finalización, Lambda envía un evento Shutdown a cada extensión externa.

Example de agregar una extensión externa a una imagen base de Python

```
FROM public.ecr.aws/lambda/python:3.11

# Copy and install the app
COPY /app /app
WORKDIR /app
RUN pip install -r requirements.txt

# Add an extension from the local directory into /opt
ADD my-extension.zip /opt
CMD python ./my-function.py
```

## Siguientes pasos

Para obtener más información acerca de las extensiones, recomendamos los siguientes recursos:

- Para obtener un ejemplo de trabajo básico, consulte [Creación de extensiones para AWS Lambda](#) en el blog de informática de AWS.
- Para obtener información acerca de las extensiones que proporcionan los socios de AWS Lambda, consulte [Introducción de extensiones de AWS Lambda](#) en el blog de informática de AWS.
- Para ver los ejemplos de extensiones y scripts envoltorios disponibles, consulte [AWS LambdaExtensiones](#) en el GitHub repositorio de AWS muestras.

## Socios de extensiones de AWS Lambda

AWS Lambda se ha asociado con varias entidades de terceros para proporcionar extensiones que se pueden integrar con las funciones de Lambda. En la lista siguiente se detallan las extensiones de terceros que están preparadas para poderlas utilizar en cualquier momento.

- [AppDynamics](#): proporciona instrumentación automática de las funciones de Lambda de Python o Node.js, para ofrecer visibilidad y alertas sobre el rendimiento de las funciones.
- [Axiom](#): proporciona paneles para supervisar el rendimiento de la función de Lambda y agregar métricas a nivel de sistema.
- [Check Point CloudGuard](#): una solución de tiempo de ejecución basada en extensiones que ofrece seguridad en todo el ciclo de vida de las aplicaciones sin servidor.
- [Datadog](#): proporciona una visibilidad completa y en tiempo real de las aplicaciones sin servidor mediante el uso de métricas, seguimientos y registros.
- [Dynatrace](#): proporciona visibilidad de seguimientos y métricas y aprovecha la IA para la detección automatizada de errores y el análisis de causa raíz en toda la pila de aplicaciones.
- [Elastic](#): proporciona supervisión del rendimiento de las aplicaciones (APM) para identificar y resolver los problemas de causa raíz mediante seguimiento, métricas y registros correlacionados.
- [Epsagon](#): escucha eventos de invocación, almacena seguimientos y los envía en paralelo a las ejecuciones de las funciones de Lambda.
- [Fastly](#): protege las funciones de Lambda de actividades sospechosas, como ataques tipo inyección, apropiación de cuentas mediante relleno de credenciales, bots maliciosos y abuso de API.
- [HashiCorp Vault](#): administra secretos y los pone a disposición de los desarrolladores para que los utilicen en código de funciones, sin comunicárselo al almacén de las funciones.
- [Honeycomb](#): herramienta de observabilidad para depurar la pila de aplicaciones.
- [Lumigo](#): crea perfiles de las invocaciones de las funciones de Lambda y recopila métricas para solucionar problemas en entornos sin servidor y de microservicios.
- [New Relic](#): se ejecuta junto con las funciones de Lambda, recopilando, mejorando y transportando de manera automática la telemetría a la plataforma de observabilidad unificada de New Relic.
- [Sedai](#): una plataforma autónoma de administración en la nube con tecnología de inteligencia artificial y machine learning, que ofrece una optimización continua para que los equipos de operaciones en la nube maximicen los ahorros de costos, el rendimiento y la disponibilidad de la nube a escala.

- [Sentry](#): diagnostica, corrige y optimiza el rendimiento de las funciones de Lambda.
- [Site24x7](#): consigue observabilidad en tiempo real en los entornos de Lambda.
- [Splunk](#): recopila métricas de alta resolución y baja latencia para ofrecer una monitorización eficiente y efectiva de las funciones de Lambda.
- [Sumo Logic](#): proporciona visibilidad del estado y el rendimiento de aplicaciones sin servidor.
- [Thundra](#): proporciona informes de telemetría asíncrona, tales como seguimientos, métricas y registros.
- [Salt Security](#): simplifica la gobernanza de la postura de la API y la seguridad de la API para las funciones de Lambda mediante la configuración automatizada y la compatibilidad con diversos tiempos de ejecución.

## Extensiones administradas de AWS

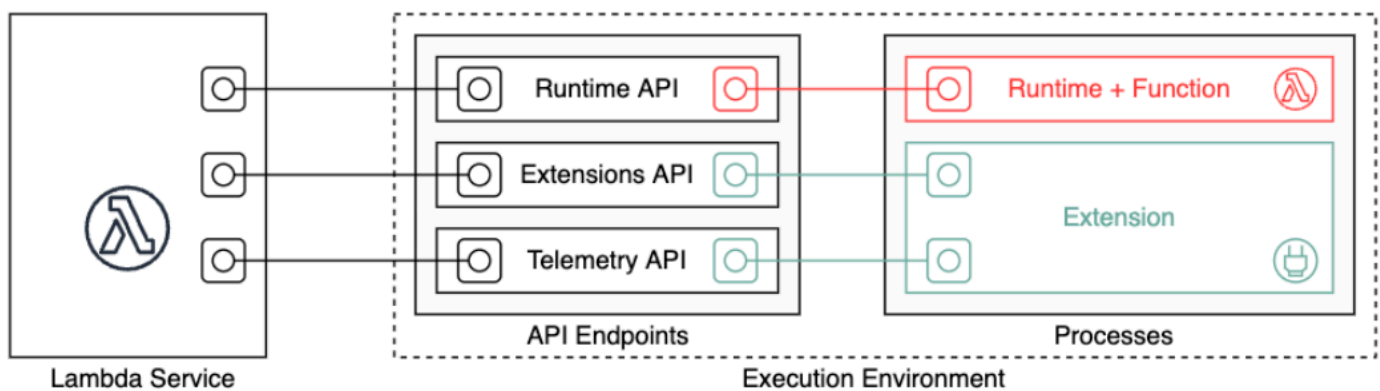
AWS proporciona sus propias extensiones administradas, como:

- [AWS AppConfig](#): utilice marcas de características y datos dinámicos para actualizar las funciones de Lambda. También puede utilizar esta extensión para actualizar otras configuraciones dinámicas, como la limitación y el ajuste de operaciones.
- [Amazon CodeGuru Profiler](#): mejora el rendimiento de las aplicaciones y reduce los costos identificando la línea de código más cara de una aplicación y ofreciendo recomendaciones para mejorar el código.
- [CloudWatch Lambda Insights](#): supervise, solucione problemas y optimice el rendimiento de las funciones de Lambda a través de paneles automatizados.
- [AWS Distro para OpenTelemetry \(ADOT\)](#): permite que las funciones envíen datos de seguimiento a servicios de monitoreo de AWS, como AWS X-Ray, así como a destinos compatibles con OpenTelemetry, tales como Honeycomb y Lightstep.
- Parámetros y secretos de AWS: permite que los clientes recuperen de manera segura parámetros del [Almacén de parámetros de AWS Systems Manager](#) y secretos de [AWS Secrets Manager](#).

Para ver más ejemplos de extensiones y proyectos de demostración, consulte [Extensiones de AWS Lambda](#).

# Utilice la API de las extensiones de Lambda para crear extensiones

Los autores de funciones de Lambda utilizan extensiones para integrar Lambda con sus herramientas preferidas de monitoreo, observabilidad, seguridad y gobernanza. Los autores de funciones pueden utilizar extensiones de AWS o de [socios de AWS](#), y proyectos de código abierto. Para obtener más información, consulte [Introducción de extensiones de AWS Lambda](#) en el blog de informática de AWS. En esta sección, se describe cómo utilizar la API de extensiones de Lambda, el ciclo de vida del entorno de ejecución de Lambda y la referencia de la API de extensiones de Lambda.



Como autor de extensiones, puede usar la API de extensiones de Lambda para integrar profundamente en el [entorno de ejecución](#) de Lambda. Su extensión puede registrarse para eventos de ciclo de vida del entorno de ejecución y función. En respuesta a estos eventos, puede iniciar nuevos procesos, ejecutar lógica, controlar y participar en todas las fases del ciclo de vida de Lambda: inicialización, invocación y apagado. Además, puede utilizar la [API de registros de tiempo de ejecución](#) para recibir una secuencia de registros.

Una extensión se ejecuta como un proceso independiente en el entorno de ejecución y puede continuar ejecutándose luego de que la invocación de la función se procese completamente. Debido a que las extensiones se ejecutan como procesos, se pueden escribir en un lenguaje diferente al de la función. Recomendamos al usuario que implemente extensiones mediante un lenguaje compilado. En este caso, la extensión es un binario autónomo compatible con los tiempos de ejecución admitidos. Todas las extensiones de soporte de [Tiempos de ejecución de Lambda](#). Si utiliza un lenguaje no compilado, asegúrese de incluir un tiempo de ejecución compatible en la extensión.

Lambda también es compatible con extensiones internas. Una extensión interna se ejecuta como un subproceso independiente en el proceso de tiempo de ejecución. El tiempo de ejecución inicia y

detiene la extensión interna. Una forma alternativa de integrarse con el entorno de Lambda es utilizar [variables de entorno específicas del lenguaje y scripts envolventes](#). Puede usarlas para configurar el entorno de tiempo de ejecución y modificar el comportamiento de inicio del proceso de tiempo de ejecución.

Puede agregar extensiones a una función de dos maneras. Para una función implementada como un [archivo .zip](#), debe implementar la extensión como una [capa](#). Para una función definida como una imagen de contenedor, se agregan [las extensiones](#) a la imagen de contenedor.

### Note

Para obtener, por ejemplo, extensiones y secuencias de comandos de envoltura, consulte [Extensiones AWS Lambda](#) en el repositorio de AWS Samples GitHub.

## Temas

- [Ciclo de vida del entorno de ejecución de Lambda](#)
- [Referencia de la API de extensiones](#)

## Ciclo de vida del entorno de ejecución de Lambda

El ciclo de vida del entorno de ejecución incluye las siguientes fases:

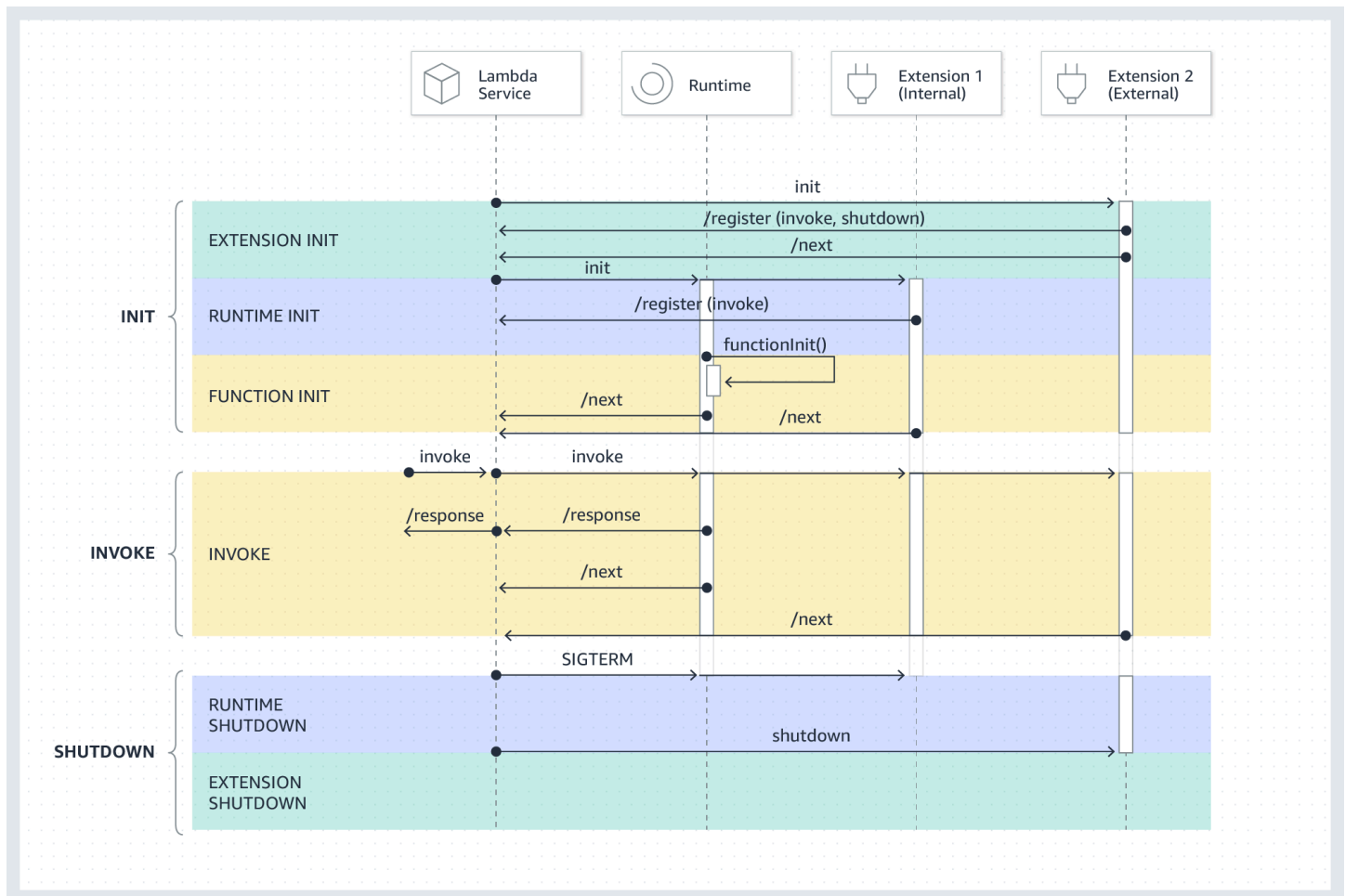
- **Init:** durante esta fase, Lambda crea o descongela un entorno de ejecución con los recursos que ha configurado, descarga el código de función y todas las capas, inicializa las extensiones, inicializa el tiempo de ejecución y ejecuta el código de inicialización de la función (el código fuera del controlador principal). La fase Init ocurre durante la primera invocación, o antes de las invocaciones de función si ha habilitado [simultaneidad aprovisionada](#).

La fase Init se divide en tres subfases: `Extension init`, `Runtime init` y `Function init`. Estas subfases garantizan que todas las extensiones y el tiempo de ejecución completen sus tareas de configuración antes de que se ejecute el código de función.

- **Invoke:** en esta fase, Lambda invoca el controlador de funciones. Después de que la función se ejecuta hasta su finalización, Lambda se prepara para manejar otra invocación de función.
- **Shutdown:** esta fase se activa si la función Lambda no recibe ninguna invocación durante un período de tiempo. En la fase Shutdown, Lambda apaga el tiempo de ejecución, alerta a las extensiones para que se detengan limpiamente y, a continuación, elimina el entorno. Lambda

envía un evento Shutdown a cada extensión, lo que indica a la extensión que el entorno está a punto de cerrarse.

Cada fase comienza con un evento desde Lambda hasta el tiempo de ejecución y en todas las extensiones registradas. El tiempo de ejecución y la finalización de cada señal de extensión mediante el envío de un solicitud de API de Next. Lambda congela el entorno de ejecución cuando cada proceso se ha completado y no hay eventos pendientes.



## Temas

- [Fase "init"](#)
- [Fase "invoke"](#)
- [Fase "shutdown"](#)
- [Permisos y configuración](#)
- [Administración de errores](#)



- [Extensiones de solución de problemas](#)

## Fase "init"

Durante la fase `Extension init`, cada extensión debe registrarse con Lambda para recibir eventos. Lambda utiliza el nombre de archivo completo de la extensión para validar que la extensión ha completado la secuencia de arranque. Por lo tanto, cada llamada `Register` a la API debe incluir el encabezado `Lambda-Extension-Name` con el nombre de archivo completo de la extensión.

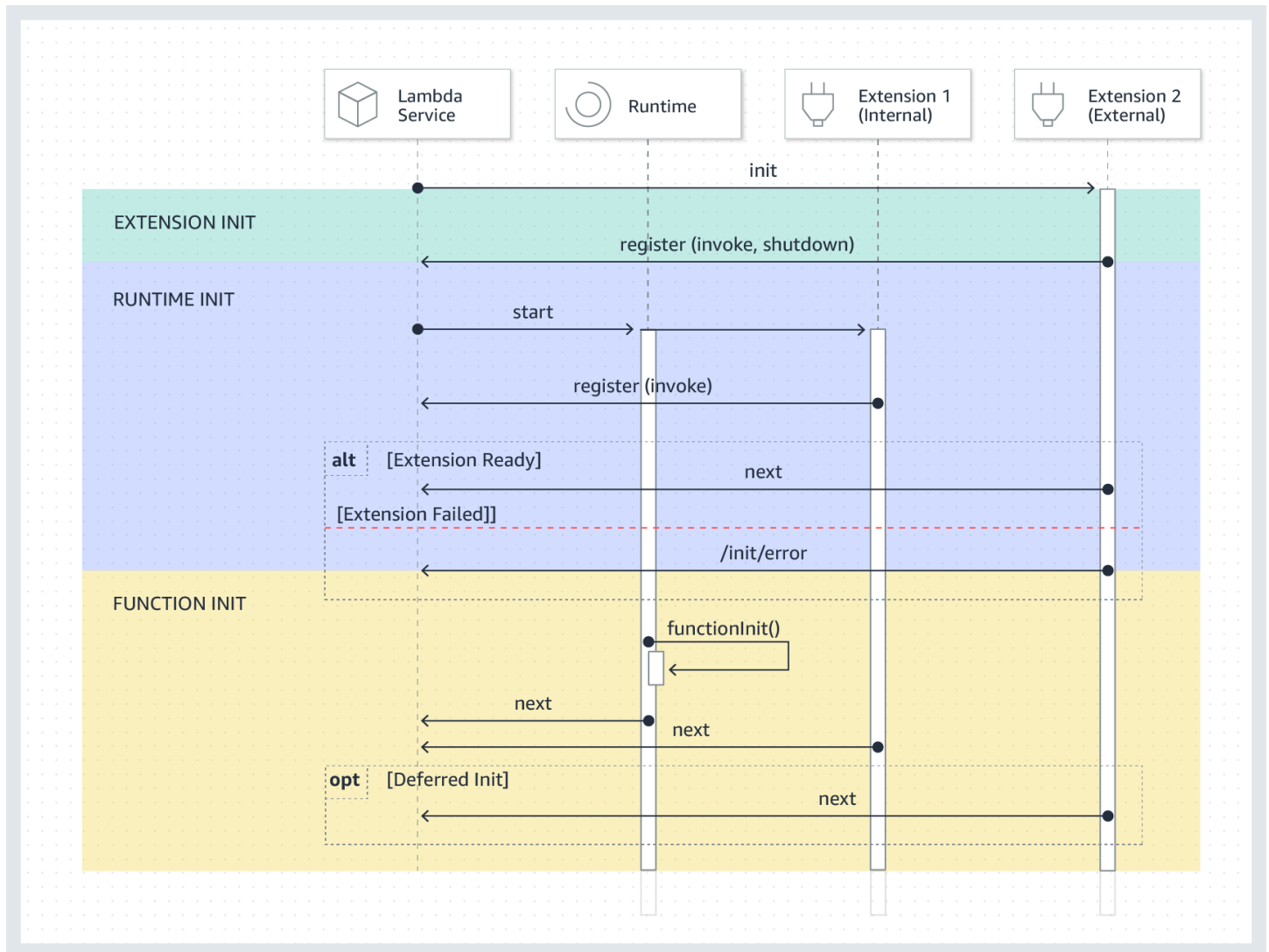
Puede registrar hasta 10 extensiones para una función. Este límite se aplica a través de la llamada `Register` a la API.

Después de que cada extensión se registre, Lambda comienza la fase `Runtime init`. El proceso en tiempo de ejecución llama `functionInit` para iniciar la fase `Function init`.

La fase `Init` se completa después del tiempo de ejecución y cada extensión registrada indica la finalización mediante el envío de una solicitud `Next` a la API.

### Note

Las extensiones pueden completar su inicialización en cualquier punto de la fase `Init`.



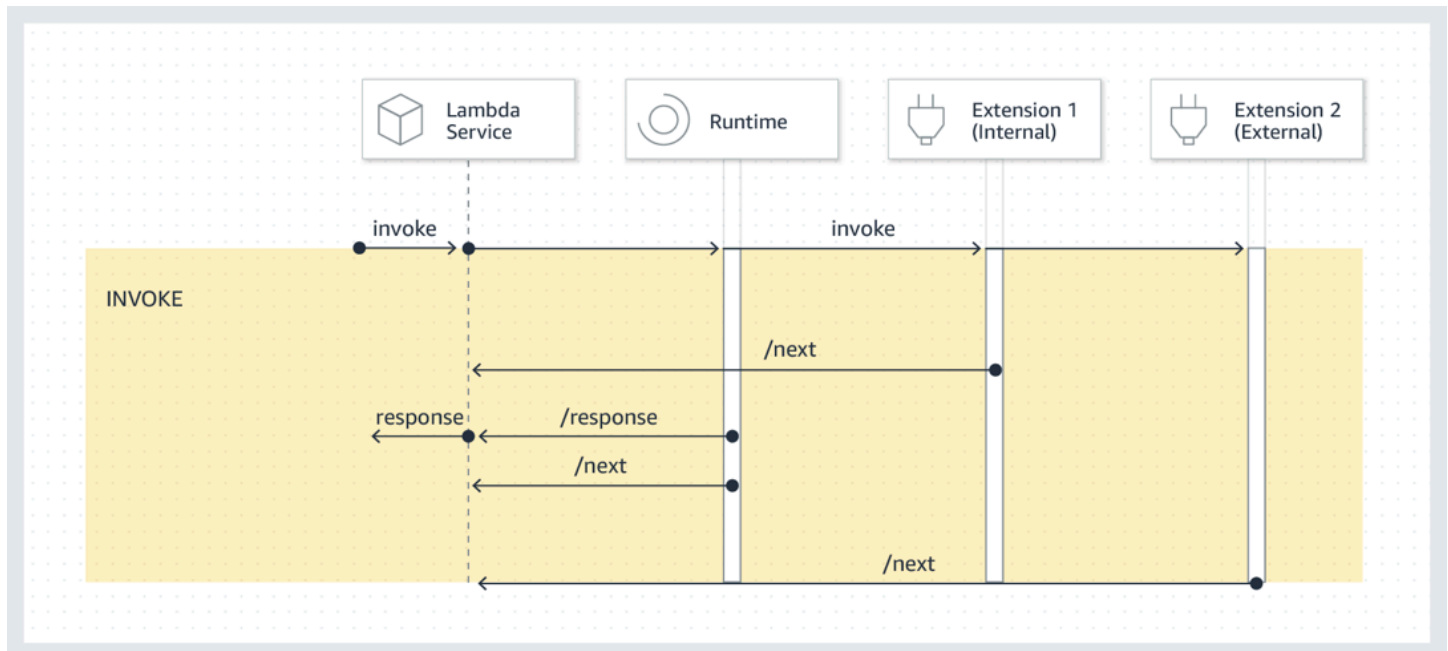
## Fase "invoke"

Cuando se invoca una función de Lambda en respuesta a una solicitud a la API Next, Lambda envía un evento `Invoke` al tiempo de ejecución y a cada extensión registrada para el evento `Invoke`.

Durante la invocación, las extensiones externas se ejecutan en paralelo a la función. También continúan ejecutándose después de que la función se haya completado. Esto permite al usuario capturar información de diagnóstico o enviar registros, métricas y rastros a una ubicación de su elección.

Después de recibir la respuesta de la función desde el tiempo de ejecución, Lambda devuelve la respuesta al cliente, incluso si las extensiones todavía se ejecutan.

La fase Invoke finaliza después de que el tiempo de ejecución y todas las extensiones indiquen que se han terminado mediante el envío de una solicitud Next a la API.



Carga útil del evento: el evento enviado al tiempo de ejecución (y a la función de Lambda) lleva toda la solicitud, encabezados (como RequestId) y carga útil. El evento enviado a cada extensión contiene metadatos que describen el contenido del evento. Este evento de ciclo de vida incluye el tipo del evento, el tiempo de espera de la función (deadlineMs), el requestId, el nombre de recurso de Amazon (ARN) de la función invocada y los encabezados de seguimiento.

Las extensiones que desean acceder al cuerpo del evento de función pueden usar un SDK en tiempo de ejecución que se comunica con la extensión. Los desarrolladores de funciones utilizan el SDK en tiempo de ejecución para enviar la carga a la extensión cuando se invoca la función.

A continuación, se muestra un ejemplo de carga:

```
{
  "eventType": "INVOKE",
  "deadlineMs": 676051,
  "requestId": "3da1f2dc-3222-475e-9205-e2e6c6318895",
  "invokedFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:ExtensionTest",
  "tracing": {
    "type": "X-Amzn-Trace-Id",
    "value":
      "Root=1-5f35ae12-0c0fec141ab77a00bc047aa2;Parent=2be948a625588e32;Sampled=1"
  }
}
```

```
}  
}
```

**Límite de duración:** la configuración de tiempo de espera de la función limita la duración de la fase Invoke al completo. Por ejemplo, si establece el tiempo de espera de la función en 360 segundos, la función y todas las extensiones deben completarse en 360 segundos. Tenga en cuenta que no hay una fase posterior a "invoke" independiente. La duración es el tiempo total que tarda su tiempo de ejecución y todas las invocaciones de sus extensiones en completarse y no se calcula hasta que la función y todas las extensiones hayan terminado de ejecutarse.

**Impacto de desempeño y sobrecarga de extensiones:** las extensiones pueden afectar al rendimiento de la función. Como autor de la extensión, tiene control sobre el impacto de desempeño de la extensión. Por ejemplo, si la extensión realiza operaciones de procesamiento intensivo, la duración de la función aumentará porque la extensión y el código de la función comparten los mismos recursos de la CPU. Además, si la extensión realiza numerosas operaciones después de completar la invocación de la función, la duración de la ejecución de la función aumentará porque la fase Invoke continuará hasta que todas las extensiones indiquen que se han completado.

#### Note

Lambda asigna la potencia de la CPU en proporción a la configuración de memoria de la función. Es posible que vea una mayor duración de ejecución e inicialización en configuraciones de memoria más bajas porque los procesos de función y extensión compiten por los mismos recursos de la CPU. Para reducir la duración de ejecución e inicialización, intente aumentar la configuración de memoria.

Para ayudar a identificar el impacto del rendimiento introducido por las extensiones de la fase Invoke, Lambda genera la métrica `PostRuntimeExtensionsDuration`. Esta métrica mide el tiempo acumulado invertido entre la solicitud Next a la API de tiempo de ejecución y la última solicitud Next a la API de la extensión. Para medir el aumento de memoria utilizada, utilice la métrica `MaxMemoryUsed`. Para obtener más información acerca de las métricas de función, consulte [Ver las métricas de funciones de Lambda](#).

Los desarrolladores de funciones pueden ejecutar diferentes versiones de sus funciones en paralelo para conocer el impacto de una extensión específica. Recomendamos que los autores de extensiones publiquen el consumo de recursos esperado para facilitar a los desarrolladores de funciones la elección de una extensión adecuada.

## Fase "shutdown"

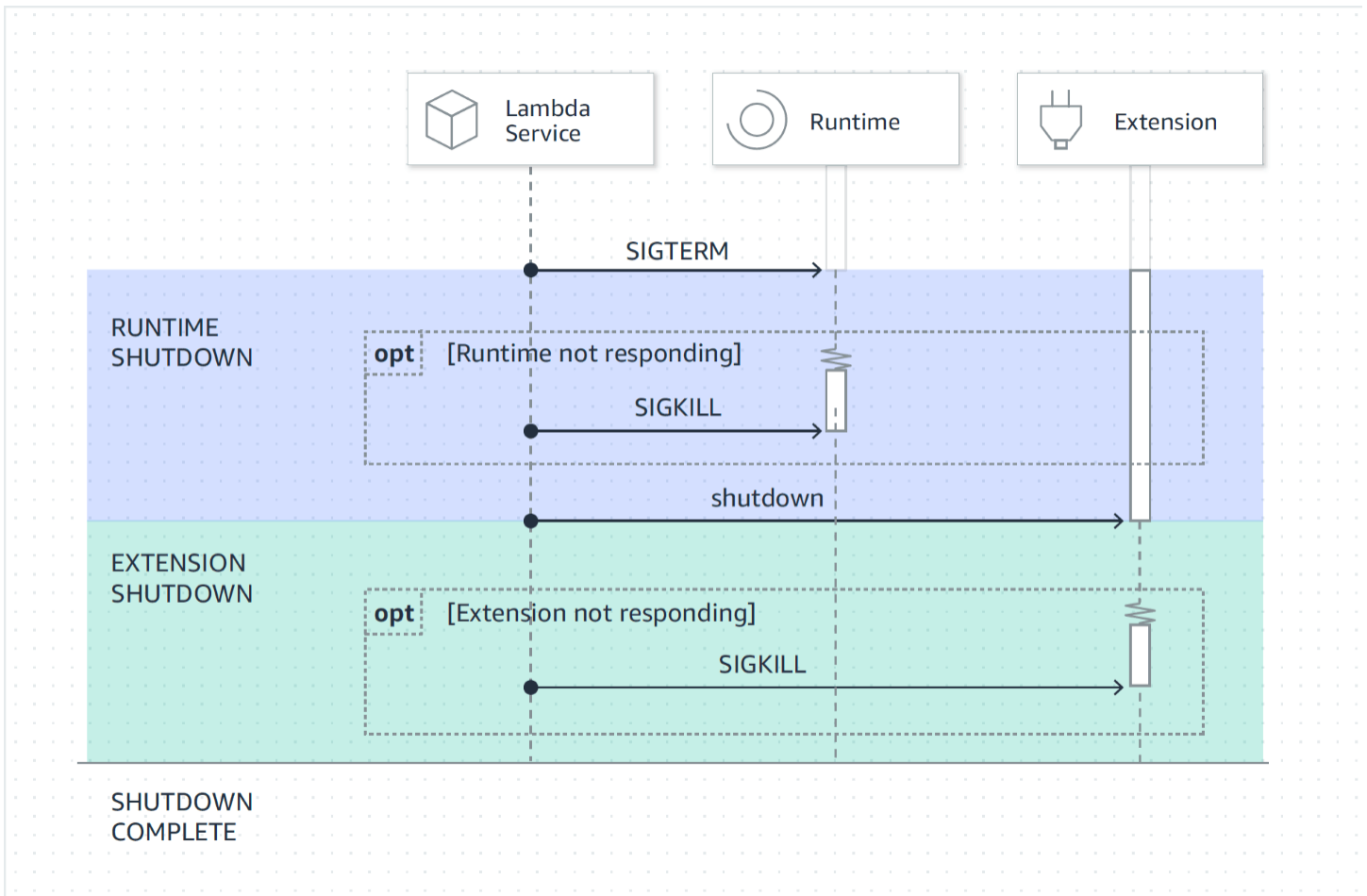
Cuando Lambda está a punto de cerrar el tiempo de ejecución, envía Shutdown a cada extensión externa registrada. Las extensiones pueden utilizar este tiempo para las tareas de limpieza finales. El evento Shutdown se envía en respuesta a una solicitud Next a la API.

Límite de duración: la duración máxima de la fase Shutdown depende de la configuración de las extensiones registradas:

- 0 ms: una función sin extensiones registradas
- 500 ms: una función con una extensión interna registrada.
- 2000 ms: una función con una o más extensiones externas registradas.

Para una función con extensiones externas, Lambda reserva hasta 300 ms (500 ms para un tiempo de ejecución con una extensión interna) para que el proceso de tiempo de ejecución realice un apagado estable. Lambda asigna el resto del límite de 2000 ms para que las extensiones externas se cierren.

Si el tiempo de ejecución o una extensión no responde al evento Shutdown dentro del límite, Lambda termina el proceso mediante una señal SIGKILL.



Carga del evento: el evento Shutdown contiene el motivo del apagado y el tiempo restante en milisegundos.

`shutdownReason` incluye los valores que se muestran a continuación:

- `SPINDOWN`: apagado normal
- `TIMEOUT`: tiempo de espera límite de duración
- `FAILURE`: condición de error, como un evento de `out-of-memory`

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time milliseconds"
}
```

## Permisos y configuración

Las extensiones se ejecutan en el mismo entorno de ejecución que la función de Lambda. Las extensiones también comparten recursos con la función, como CPU, memoria y almacenamiento en disco /tmp. Además, las extensiones utilizan el mismo rol de AWS Identity and Access Management (IAM) y el mismo contexto de seguridad que la función.

Permisos de acceso al sistema de archivos y a la red: las extensiones se ejecutan en el mismo espacio de nombres de sistema de archivos y nombre de red que el tiempo de ejecución de la función. Esto significa que las extensiones deben ser compatibles con el sistema operativo asociado. Si una extensión requiere reglas de tráfico de salida de red adicionales, se deben aplicar estas reglas a la configuración de la función.

### Note

Debido a que el directorio de código de función es de solo lectura, las extensiones no pueden modificar el código de función.

Variables de entorno: las extensiones pueden acceder a las [variables de entorno](#) de la función, excepto las siguientes variables que son específicas del proceso de tiempo de ejecución:

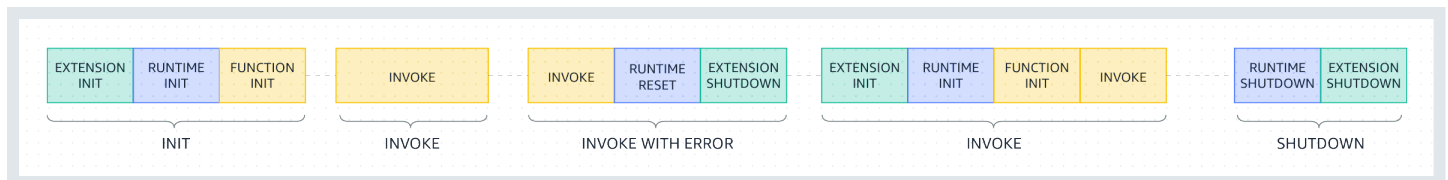
- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

## Administración de errores

Errores de inicialización: si se produce un error en una extensión, Lambda reinicia el entorno de ejecución para imponer un comportamiento consistente y fomentar el error rápido para las extensiones. Además, para algunos clientes, las extensiones deben satisfacer necesidades críticas como registro, seguridad, gobierno y recopilación de telemetría.

Errores de Invoke (como la falta de memoria, el tiempo de espera de la función): dado que las extensiones comparten recursos con el tiempo de ejecución, el agotamiento de la memoria los afecta. Cuando el tiempo de ejecución produce un error, todas las extensiones y el propio tiempo de ejecución participan en la fase Shutdown. Además, el tiempo de ejecución se reinicia automáticamente como parte de la invocación actual o a través de un mecanismo de reinicialización diferido.

Si hay un error (como un tiempo de espera de función o un error de tiempo de ejecución) durante Invoke, el servicio de Lambda realiza un restablecimiento. El restablecimiento se comporta como un evento Shutdown. Primero, Lambda apaga el tiempo de ejecución, luego envía un evento Shutdown a cada extensión externa registrada. El evento incluye el motivo del apagado. Si este entorno se utiliza para una nueva invocación, la extensión y el tiempo de ejecución se vuelven a inicializar como parte de la siguiente invocación.



Para obtener una explicación más detallada del diagrama anterior, consulte [Errores durante la fase Invoke](#).

Registros de extensión: Lambda envía la salida de registro de las extensiones a CloudWatch Logs. Lambda también genera un evento de registro adicional para cada extensión durante Init. El evento de registro registra el nombre y la preferencia de registro ("event", "config") en caso de éxito o el motivo del error en caso de error.

## Extensiones de solución de problemas

- Si se produce un error en una solicitud Register, asegúrese de que el encabezado de Lambda-Extension-Name de la llamada Register a la API contiene el nombre de archivo completo de la extensión.



- Si la solicitud de `Register` produce un error para una extensión interna, asegúrese de que la solicitud no se registra para el evento `Shutdown`.

## Referencia de la API de extensiones

La especificación de OpenAPI para la versión de la API de extensiones 2020-01-01 está disponible aquí: [extensions-api.zip](#)

Puede recuperar el valor del punto de enlace de la API desde la variable de entorno de `AWS_LAMBDA_RUNTIME_API`. Para enviar una solicitud de `Register`, utilice el prefijo `2020-01-01/` antes de cada ruta de la API. Por ejemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

### Métodos de API

- [Regístrese](#)
- [Next](#)
- [Error de init](#)
- [Error de salida](#)

## Regístrese

Durante `Extension init`, todas las extensiones deben registrarse en Lambda para recibir eventos. Lambda utiliza el nombre de archivo completo de la extensión para validar que la extensión ha completado la secuencia de arranque. Por lo tanto, cada llamada `Register` a la API debe incluir el encabezado `Lambda-Extension-Name` con el nombre de archivo completo de la extensión.

Las extensiones internas comienzan y paran según el proceso de tiempo de ejecución, por lo que no se les permite registrarse para el evento `Shutdown`.

Ruta – `/extension/register`

Método: `POST`

### Encabezados de solicitudes

- `Lambda-Extension-Name`: el nombre de archivo completo de la extensión. Requerido: sí. Tipo: cadena.

- **Lambda-Extension-Accept-Feature**: utilice esta opción para especificar las funciones opcionales de las extensiones durante el registro. Obligatorio: no Tipo: cadena separada por comas. Funciones disponibles para especificar mediante esta configuración:
  - **accountId**: si se especifica, la respuesta de registro de la extensión contendrá el ID de cuenta asociado a la función de Lambda para la que está registrando la extensión.

### Parámetros del cuerpo de la solicitud

- **events**: matriz de los eventos para registrarse. Obligatorio: no Tipo: matriz de cadenas. Cadenas válidas: INVOKE y SHUTDOWN.

### Encabezados de respuesta

- **Lambda-Extension-Identifier**: identificador de agente único generado (cadena UUID) que se requiere para todas las solicitudes posteriores.

### Códigos de respuesta

- **200**: el cuerpo de la respuesta contiene el nombre de la función, la versión de la función y el nombre del controlador.
- **400**: solicitud errónea
- **403**: prohibido
- **500**: error en contenedor. Estado no recuperable. La extensión debe salir rápidamente.

### Example Ejemplo de cuerpo de solicitud

```
{
  'events': [ 'INVOKE', 'SHUTDOWN' ]
}
```

### Example Ejemplo de cuerpo de respuesta

```
{
  "functionName": "helloWorld",
  "functionVersion": "$LATEST",
  "handler": "lambda_function.lambda_handler"
```

```
}
```

### Example Ejemplo de cuerpo de respuesta con función accountId opcional

```
{  
  "functionName": "helloWorld",  
  "functionVersion": "$LATEST",  
  "handler": "lambda_function.lambda_handler",  
  "accountId": "123456789012"  
}
```

## Next

Las extensiones envían una solicitud Next a la API para recibir el siguiente evento, que puede ser un evento Invoke o un evento Shutdown. El cuerpo de la respuesta contiene la carga, que es un documento JSON que contiene datos de eventos.

La extensión envía una solicitud Next a la API para indicar que está lista para recibir nuevos eventos. Esta es una llamada de bloqueo.

No establezca un tiempo de espera en la llamada GET, ya que la extensión puede suspenderse durante un periodo de tiempo hasta que haya un evento que devolver.

Ruta – /extension/event/next

Método: GET

Encabezados de solicitudes

- **Lambda-Extension-Identifier**: identificador único para la extensión (cadena UUID).  
Requerido: sí. Tipo: cadena UUID.

Encabezados de respuesta

- **Lambda-Extension-Event-Identifier**: identificador único para el evento (cadena UUID).

Códigos de respuesta

- **200**: la respuesta contiene información sobre el próximo evento (EventInvoke o EventShutdown).

- 403: prohibido
- 500: error en contenedor. Estado no recuperable. La extensión debe salir rápidamente.

## Error de init

La extensión utiliza este método para informar de un error de inicialización en Lambda. Llámelo cuando la extensión produce un error al inicializarse después de que se haya registrado. Después de que Lambda reciba el error, no hay más llamadas a la API correctas. La extensión debe cerrarse después de recibir la respuesta de Lambda.

Ruta – `/extension/init/error`

Método: POST

Encabezados de solicitudes

- `Lambda-Extension-Identifier`: identificador único para la extensión. Requerido: sí. Tipo: cadena UUID.
- `Lambda-Extension-Function-Error-Type`: tipo de error que encontró la extensión. Requerido: sí. Este encabezado consta de un valor de cadena. Lambda acepta cualquier cadena, pero recomendamos un formato de `<category.reason>`. Por ejemplo:
  - `Extension.NoSuchHandler`
  - `Extension.APIKeyNotFound`
  - `Extension.ConfigInvalid`
  - `Extension.UnknownReason`

Parámetros del cuerpo de la solicitud

- `ErrorRequest`: información sobre el error. Obligatorio: no

Este campo es un objeto JSON con la siguiente estructura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
  stackTrace: array of strings
}
```

Tenga en cuenta que Lambda acepta cualquier valor para `errorType`.

En el ejemplo siguiente, se muestra un mensaje de error de función de Lambda en el que la función no pudo analizar los datos de evento proporcionados en la invocación.

#### Example Error de la función

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException",
  "stackTrace": [ ]
}
```

#### Códigos de respuesta

- 202: aceptada
- 400: solicitud errónea
- 403: prohibido
- 500: error en contenedor. Estado no recuperable. La extensión debe salir rápidamente.

#### Error de salida

La extensión utiliza este método para informar de un error en Lambda antes de salir. Llámelo cuando encuentre un error inesperado. Después de que Lambda reciba el error, no hay más llamadas a la API correctas. La extensión debe cerrarse después de recibir la respuesta de Lambda.

Ruta – `/extension/exit/error`

Método: POST

#### Encabezados de solicitudes

- `Lambda-Extension-Identifier`: identificador único para la extensión. Requerido: sí. Tipo: cadena UUID.
- `Lambda-Extension-Function-Error-Type`: tipo de error que encontró la extensión. Requerido: sí. Este encabezado consta de un valor de cadena. Lambda acepta cualquier cadena, pero recomendamos un formato de `<category.reason>`. Por ejemplo:
  - `Extension.NoSuchHandler`
  - `Extension.APIKeyNotFound`

- `Extension.ConfigInvalid`
- `Extension.UnknownReason`

### Parámetros del cuerpo de la solicitud

- `ErrorRequest`: información sobre el error. Obligatorio: no

Este campo es un objeto JSON con la siguiente estructura:

```
{
  errorMessage: string (text description of the error),
  errorType: string,
  stackTrace: array of strings
}
```

Tenga en cuenta que Lambda acepta cualquier valor para `errorType`.

En el ejemplo siguiente, se muestra un mensaje de error de función de Lambda en el que la función no pudo analizar los datos de evento proporcionados en la invocación.

### Example Error de la función

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException",
  "stackTrace": [ ]
}
```

### Códigos de respuesta

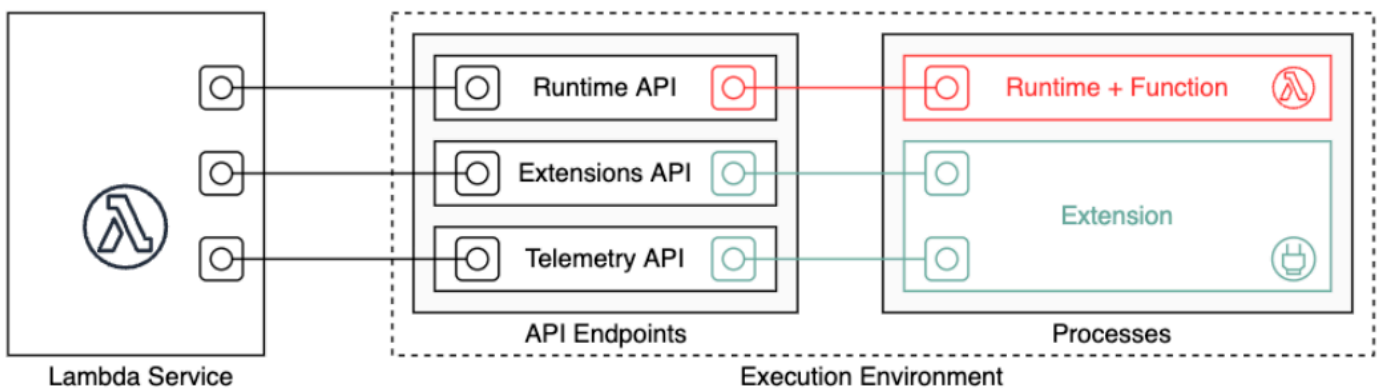
- 202: aceptada
- 400: solicitud errónea
- 403: prohibido
- 500: error en contenedor. Estado no recuperable. La extensión debe salir rápidamente.

## Acceso a datos de telemetría en tiempo real para extensiones mediante la API de telemetría

La API de telemetría de Lambda les permite a sus extensiones recibir datos de telemetría directamente de Lambda. Durante la inicialización e invocación de la función, Lambda captura automáticamente la telemetría, como los registros, las métricas de la plataforma y los seguimientos de la plataforma. La API de telemetría les permite a las extensiones obtener estos datos de telemetría directamente de Lambda casi en tiempo real.

Dentro del entorno de ejecución de Lambda, puede suscribir las extensiones de Lambda a flujos de telemetría. Tras suscribirse, Lambda envía automáticamente todos los datos de telemetría a sus extensiones. A continuación, tiene flexibilidad para procesar, filtrar y despachar los datos a su destino preferido, como un bucket de Amazon Simple Storage Service (Amazon S3) o un proveedor de herramientas de observabilidad de terceros.

El siguiente diagrama muestra cómo la API de extensiones y la API de telemetría enlazan las extensiones a Lambda desde el entorno de ejecución. Además, la API de tiempo de ejecución conecta el tiempo de ejecución y la función a Lambda.



### ⚠ Important

La API de telemetría de Lambda reemplaza a la API de registros de Lambda. Si bien la API de registros sigue siendo completamente funcional, recomendamos utilizar solo la API de telemetría en el futuro. Puede suscribir su extensión a una transmisión de telemetría mediante la API de telemetría o la API de registros. Tras suscribirse mediante una de estas API, cualquier intento de suscribirse mediante la otra API volverá a causar un error.

Las extensiones pueden utilizar la API de telemetría para suscribirse a tres flujos de telemetría diferentes:

- Telemetría de plataforma: registros, métricas y seguimientos, que describen los eventos y errores relacionados con el ciclo de vida del tiempo de ejecución del entorno de ejecución, el ciclo de vida de las extensiones y las invocaciones de funciones.
- Registros de funciones: registros personalizados que genera el código de la función de Lambda.
- Registros de extensiones: registros personalizados generados por el código de extensión de Lambda.

#### Note

Lambda envía registros y métricas a CloudWatch y seguimientos a X-Ray (si lo tiene activado), incluso si una extensión se suscribe a los flujos de telemetría.

## Secciones

- [Creación de extensiones mediante la API de telemetría](#)
- [Registro de sus extensiones](#)
- [Creación de un oyente de telemetría](#)
- [Especificación de un protocolo de destino](#)
- [Configuración del uso de memoria y el almacenamiento en búfer](#)
- [Envío de una solicitud de suscripción a la API de telemetría](#)
- [Mensajes entrantes de la API de telemetría](#)
- [Referencia de la API de telemetría de Lambda](#)
- [Referencia del esquema Event de la API de telemetría de Lambda](#)
- [Conversión de objetos Event de la API de telemetría de Lambda en OpenTelemetry Spans](#)
- [Uso de la API de registros de Lambda](#)

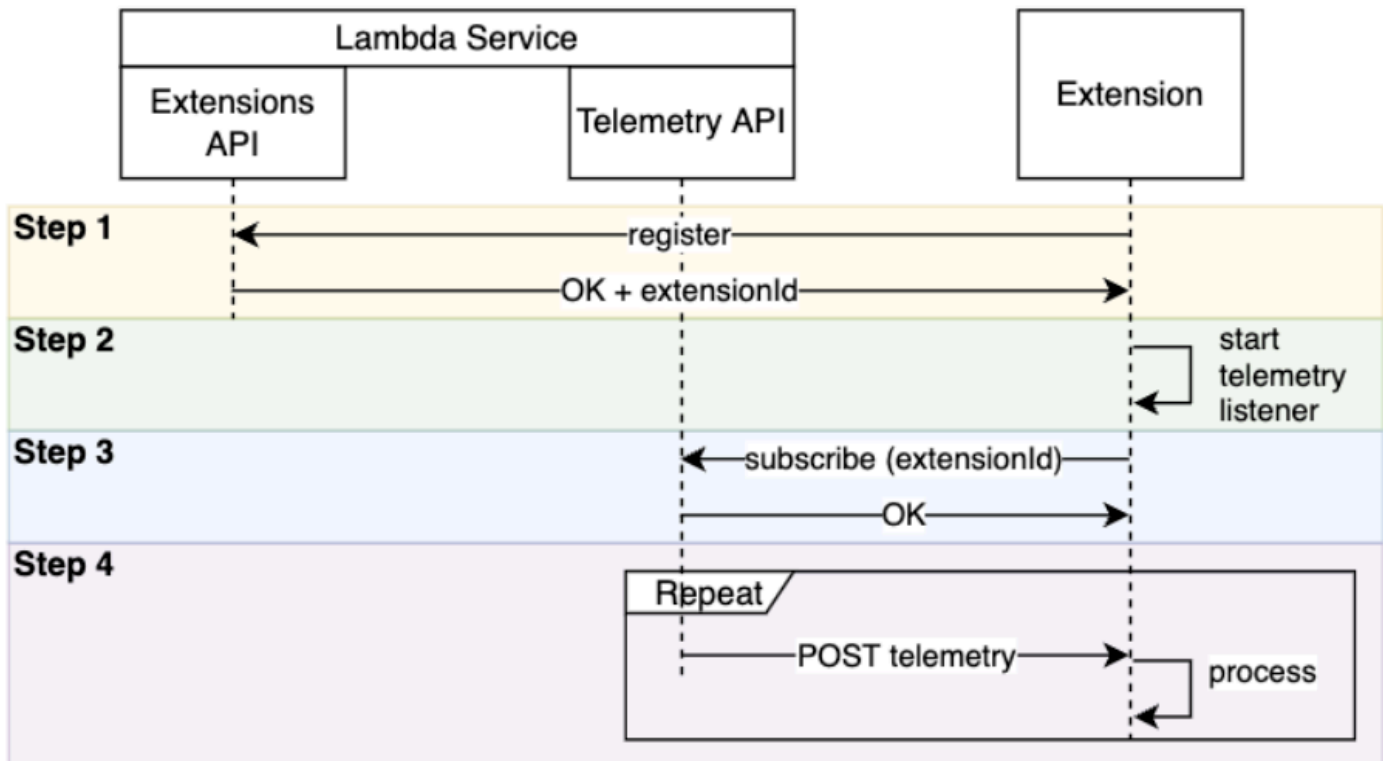
## Creación de extensiones mediante la API de telemetría

Las extensiones de Lambda se ejecutan como un proceso independiente en el entorno de ejecución. Las extensiones se pueden seguir ejecutando una vez que finaliza la invocación de la función. Dado que las extensiones son procesos separados, puede escribirlas en un idioma diferente al del código



de la función. Recomendamos al usuario que escriba extensiones mediante un lenguaje compilado como Golang o Rust. De esta manera, la extensión es un binario autónomo compatible con cualquier tiempo de ejecución admitido.

El siguiente diagrama ilustra un proceso de cuatro pasos para crear una extensión que reciba y procese datos de telemetría mediante la API de telemetría.



A continuación, se muestra cada paso en mayor profundidad:

1. Registre su extensión mediante [the section called “API de extensiones”](#). Esto le proporciona un `Lambda-Extension-Identifier`, que necesitará en los siguientes pasos. Para obtener más información sobre cómo registrar su extensión, consulte [the section called “Registro de sus extensiones”](#).
2. Cree un oyente de telemetría. Puede ser un servidor HTTP o TCP básico. Lambda usa el URI del oyente de telemetría para enviar datos de telemetría a su extensión. Para obtener más información, consulte [the section called “Creación de un oyente de telemetría”](#).
3. Mediante la API de suscripción de la API de telemetría, suscriba su extensión a los flujos de telemetría deseados. Necesitará el URI de su oyente de telemetría para este paso. Para obtener más información, consulte [the section called “Envío de una solicitud de suscripción a la API de telemetría”](#).

4. Obtenga datos de telemetría de Lambda a través del oyente de telemetría. Puede realizar cualquier procesamiento personalizado de estos datos, como enviar los datos a Amazon S3 o a un servicio de observabilidad externo.

#### Note

El entorno de ejecución de una función de Lambda puede iniciarse y detenerse varias veces como parte de su [ciclo de vida](#). En general, el código de extensión se ejecuta durante las invocaciones de funciones y también hasta dos segundos durante la fase de apagado. Se recomienda agrupar la telemetría en lotes a medida que llegue al oyente. A continuación, utilice los eventos del ciclo de vida Invoke y Shutdown para enviar cada lote a los destinos que desee.

## Registro de sus extensiones

Antes de poder suscribirse para recibir datos de telemetría, debe registrar su extensión de Lambda. El registro se produce durante la [fase de inicialización de la extensión](#). En el siguiente ejemplo se muestra una solicitud HTTP para registrar una extensión.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
Lambda-Extension-Name: lambda_extension_name
{
  'events': [ 'INVOKE', 'SHUTDOWN' ]
}
```

Si la solicitud se realiza correctamente, el suscriptor recibe una respuesta correcta HTTP 200. El encabezado de la respuesta contiene el `Lambda-Extension-Identifier`. El cuerpo de respuesta contiene otras propiedades de la función.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
{
  "functionName": "lambda_function",
  "functionVersion": "$LATEST",
  "handler": "lambda_handler",
  "accountId": "123456789012"
}
```

Para obtener más información, consulte [the section called “Referencia de la API de extensiones”](#).

## Creación de un oyente de telemetría

La extensión de Lambda debe tener un oyente que gestione las solicitudes entrantes de la API de telemetría. En el siguiente código se muestra un ejemplo de implementación de un oyente de telemetría en Golang:

```
// Starts the server in a goroutine where the log events will be sent
func (s *TelemetryApiListener) Start() (string, error) {
    address := listenOnAddress()
    l.Info("[listener:Start] Starting on address", address)
    s.httpServer = &http.Server{Addr: address}
    http.HandleFunc("/", s.http_handler)
    go func() {
        err := s.httpServer.ListenAndServe()
        if err != http.ErrServerClosed {
            l.Error("[listener:goroutine] Unexpected stop on Http Server:", err)
            s.Shutdown()
        } else {
            l.Info("[listener:goroutine] Http Server closed:", err)
        }
    }()
    return fmt.Sprintf("http://%s/", address), nil
}

// http_handler handles the requests coming from the Telemetry API.
// Everytime Telemetry API sends log events, this function will read them from the
// response body
// and put into a synchronous queue to be dispatched later.
// Logging or printing besides the error cases below is not recommended if you have
// subscribed to
// receive extension logs. Otherwise, logging here will cause Telemetry API to send new
// logs for
// the printed lines which may create an infinite loop.
func (s *TelemetryApiListener) http_handler(w http.ResponseWriter, r *http.Request) {
    body, err := ioutil.ReadAll(r.Body)
    if err != nil {
        l.Error("[listener:http_handler] Error reading body:", err)
        return
    }

    // Parse and put the log messages into the queue
```

```
var slice []interface{}
_ = json.Unmarshal(body, &slice)

for _, el := range slice {
    s.LogEventsQueue.Put(el)
}

l.Info("[listener:http_handler] logEvents received:", len(slice), " LogEventsQueue
length:", s.LogEventsQueue.Len())
slice = nil
}
```

## Especificación de un protocolo de destino

Al suscribirse para recibir telemetría mediante la API de telemetría, puede especificar un protocolo de destino además del URI de destino:

```
{
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
  }
}
```

Lambda acepta dos protocolos para recibir telemetría:

- HTTP (recomendado): Lambda entrega telemetría a un punto de conexión HTTP local (`http://sandbox.localdomain:${PORT}/${PATH}`) como una matriz de registros en formato JSON. El parámetro `$PATH` es opcional. Lambda solo admite HTTP, no HTTPS. Lambda ofrece telemetría a través de solicitudes POST.
- TCP: Lambda entrega telemetría a un puerto TCP en [formato JSON delimitado por línea nueva \(NDJSON\)](#).

### Note

Recomendamos encarecidamente utilizar HTTP en lugar de TCP. Con TCP, la plataforma de Lambda no puede reconocer cuando la telemetría se entrega a la capa de aplicación. Por lo tanto, si su extensión se bloquea, puede perder la telemetría. HTTP no tiene esta limitación.

Antes de suscribirse para recibir telemetría, establezca el oyente HTTP local o el puerto TCP.

Durante la instalación, tenga en cuenta lo siguiente:

- Lambda envía telemetría solo a destinos que se encuentran dentro del entorno de ejecución.
- Lambda vuelve a intentar enviar telemetría (con retroceso) si no hay ningún oyente o si la solicitud POST encuentra un error. Si el oyente de telemetría se bloquea, continuará recibiendo telemetría después de que Lambda reinicie el entorno de ejecución.
- Lambda reserva el puerto 9001. No hay otras restricciones ni recomendaciones sobre el número de puerto.

## Configuración del uso de memoria y el almacenamiento en búfer

El uso de memoria en un entorno de ejecución aumenta de forma lineal con el número de suscriptores. Las suscripciones consumen recursos de memoria porque cada una abre un nuevo búfer de memoria para almacenar datos de telemetría. El uso de memoria del búfer cuenta para el consumo general de memoria en el entorno de ejecución.

Al suscribirse para recibir telemetría mediante la API de telemetría, tiene la opción de almacenar en búfer los datos de telemetría y entregarlos a los suscriptores en lotes. Para optimizar el uso de memoria, puede especificar la configuración de almacenamiento en búfer:

```
{
  "buffering": {
    "maxBytes": 256*1024,
    "maxItems": 1000,
    "timeoutMs": 100
  }
}
```

Parámetro	Descripción	Valores predeterminados y límites
maxBytes	El volumen máximo de telemetría (en bytes) que se debe almacenar en memoria.	Predeterminado: 262 144 Mínimo: 262 144 Máximo: 1 048 576

Parámetro	Descripción	Valores predeterminados y límites
<code>maxItems</code>	El número máximo de eventos que se deben almacenar en memoria.	Predeterminado: 10 000 Mínimo: 1000 Máximo: 10 000
<code>timeoutMs</code>	Tiempo máximo (en milisegundos) para almacenar en búfer un lote.	Predeterminado: 1000 Mínimo: 25 Máximo: 30 000

Cuando configure el almacenamiento en búfer, tenga en cuenta lo siguiente:

- Si alguno de los flujos de entrada está cerrado, Lambda vacía los registros. Esto puede suceder si, por ejemplo, se bloquea el tiempo de ejecución.
- Cada suscriptor puede personalizar su configuración de almacenamiento en búfer en la solicitud de suscripción.
- Cuando determine el tamaño del búfer para leer los datos, prevea recibir cargas tan grandes como  $2 * \text{maxBytes} + \text{metadataBytes}$ , donde `maxBytes` es un componente de su configuración de almacenamiento en búfer. Para calcular la cantidad de `metadataBytes` a tener en cuenta, revise los siguientes metadatos. Lambda agrega apéndices de metadatos similares a este a cada registro:

```
{
  "time": "2022-08-20T12:31:32.123Z",
  "type": "function",
  "record": "Hello World"
}
```

- Si el suscriptor no puede procesar la telemetría entrante con la suficiente rapidez o su código de función genera un volumen muy alto de registro, Lambda podría soltar registros para mantener limitada la utilización de la memoria. Cuando esto ocurre, Lambda envía un evento `platform.logsDropped`.

## Envío de una solicitud de suscripción a la API de telemetría

Las extensiones de Lambda pueden suscribirse para recibir datos de telemetría enviando una solicitud de suscripción a la API de telemetría. La solicitud de suscripción debe contener información sobre los tipos de eventos a los que quiere que se suscriba la extensión. Además, la solicitud puede contener [información sobre el destino de la entrega](#) y una [configuración de almacenamiento en búfer](#).

Antes de enviar una solicitud de suscripción, debe tener un ID de extensión (Lambda-Extension-Identfier). Cuando [registra su extensión con la API de extensiones](#), obtiene un ID de extensión de la respuesta de la API.

La suscripción se produce durante la [fase de inicialización de la extensión](#). El siguiente ejemplo muestra una solicitud HTTP para suscribirse a los tres flujos de telemetría: telemetría de plataforma, registros de funciones y registros de extensiones.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
  "schemaVersion": "2022-12-13",
  "types": [
    "platform",
    "function",
    "extension"
  ],
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
  }
}
```

Si la solicitud se realiza correctamente, entonces el suscriptor recibe una respuesta correcta HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

## Mensajes entrantes de la API de telemetría

Tras suscribirse mediante la API de telemetría, una extensión comienza a recibir automáticamente la telemetría de Lambda mediante solicitudes POST. El cuerpo de cada solicitud POST contiene una matriz de objetos Event. Cada Event tiene el siguiente esquema:

```
{
  time: String,
  type: String,
  record: Object
}
```

- La propiedad `time` define cuándo la plataforma Lambda generó el evento. Esto difiere de cuando ocurrió realmente el evento. El valor de la cadena de `time` es una marca de tiempo en formato ISO 8601.
- La propiedad `type` define el tipo de evento. En la siguiente table se describen todos los posibles valores.
- La propiedad `record` define un objeto JSON que contiene los datos de telemetría. El esquema de este objeto JSON depende del `type`.

La siguiente tabla resume todos los tipos de objetos de Event y se enlaza con la [referencia del Event del esquema de la API de telemetría](#) para cada tipo de evento.

Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Evento de plataforma	<code>platform.initStart</code>	Se inició la inicialización de la función.	esquema <a href="#">the section called “platform.initStart”</a>
Evento de plataforma	<code>platform.initRuntimeDone</code>	Se completó la inicialización de la función.	esquema <a href="#">the section called “platform.initRuntimeDone”</a>



Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Evento de plataforma	<code>platform.initReport</code>	Un informe de inicialización de la función.	esquema <a href="#">the section called “platform.initReport”</a>
Evento de plataforma	<code>platform.start</code>	Se inició la invocación de la función.	esquema <a href="#">the section called “platform.start”</a>
Evento de plataforma	<code>platform.runtimeDone</code>	El tiempo de ejecución terminó de procesar un evento con éxito o con falla.	esquema <a href="#">the section called “platform.runtimeDone”</a>
Evento de plataforma	<code>platform.report</code>	Un informe de la invocación de funciones.	esquema <a href="#">the section called “platform.report”</a>
Evento de plataforma	<code>platform.restoreStart</code>	Se inició la restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreStart”</a>
Evento de plataforma	<code>platform.restoreRuntimeDone</code>	Se completó la restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreRuntimeDone”</a>
Evento de plataforma	<code>platform.restoreReport</code>	Informe de restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreReport”</a>
Evento de plataforma	<code>platform.telemetrySubscription</code>	La extensión suscrita a la API de telemetría.	esquema <a href="#">the section called “platform.telemetrySubscription”</a>

Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Evento de plataforma	platform. logsDropped	Lambda ha eliminado las entradas de registro.	esquema <a href="#">the section called “platform.logsDropped”</a>
Registros de funciones	function	Una línea de registro del código de la función.	esquema <a href="#">the section called “function”</a>
Registros de extensión	extension	Una línea de registro del código de extensión.	esquema <a href="#">the section called “extension”</a>

## Referencia de la API de telemetría de Lambda

Utilice el punto de conexión de la API de telemetría de Lambda para suscribir extensiones a los flujos de telemetría. Puede recuperar el punto de conexión de la API de telemetría desde la variable de entorno `AWS_LAMBDA_RUNTIME_API`. Para enviar una solicitud de API, agregue la versión de la API (`2022-07-01/`) y `telemetry/`. Por ejemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Para ver la definición de la especificación OpenAPI (OAS) de la versión de respuestas de suscripción `2022-12-13`, consulte lo siguiente:

- HTTP: [telemetry-api-http-schema.zip](#)
- TCP: [.zip telemetry-api-tcp-schema](#)

### Operaciones de la API

- [Suscribirse](#)

### Suscribirse

Para suscribirse a un flujo de telemetría, una extensión de Lambda puede enviar una solicitud de API de suscripción.

- Ruta – `/telemetry`
- Método – PUT
- Encabezados
  - `Content-Type: application/json`
- Parámetros del cuerpo de la solicitud
  - `schemaVersion`
    - Obligatorio: sí
    - Tipo: cadena
    - Valores válidos: `"2022-12-13"` o `"2022-07-01"`
  - `destino`: los parámetros de configuración que definen el destino del evento de telemetría y el protocolo para la entrega del evento.
    - Obligatorio: sí

- Tipo: objeto

```
{
  "protocol": "HTTP",
  "URI": "http://sandbox.localdomain:8080"
}
```

- protocolo: el protocolo que utiliza Lambda para enviar datos de telemetría.
  - Obligatorio: sí
  - Tipo: cadena
  - Valores válidos: "HTTP"|"TCP"
- URI: el URI al que se envían los datos de telemetría.
  - Obligatorio: sí
  - Tipo: cadena
- Para obtener más información, consulte [the section called “Especificación de un protocolo de destino”](#).
- tipos: los tipos de telemetría a los que desea que se suscriba la extensión.
  - Obligatorio: sí
  - Tipo: matriz de cadenas
  - Valores válidos: "platform"|"function"|"extension"
- almacenamiento en búfer: los ajustes de configuración para el almacenamiento en búfer de eventos.
  - Obligatorio: no
  - Tipo: objeto

```
{
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  }
}
```

- maxItems: el número máximo de eventos que se deben almacenar en memoria.
  - Obligatorio: no
  - Tipo: entero

- Predeterminado: 1000
- Mínimo: 1000
- Máximo: 10 000
- maxBytes: el volumen máximo de telemetría (en bytes) que se debe almacenar en memoria.
  - Obligatorio: no
  - Tipo: entero
  - Predeterminado: 262 144
  - Mínimo: 262 144
  - Máximo: 1 048 576
- timeoutMs: el tiempo máximo (en milisegundos) para almacenar en búfer un lote.
  - Obligatorio: no
  - Tipo: entero
  - Predeterminado: 1000
  - Mínimo: 25
  - Máximo: 30 000
- Para obtener más información, consulte [the section called “Configuración del uso de memoria y el almacenamiento en búfer”](#).

### Ejemplo de solicitud de API de suscripción

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
  "schemaVersion": "2022-12-13",
  "types": [
    "platform",
    "function",
    "extension"
  ],
  "buffering": {
    "maxItems": 1000,
    "maxBytes": 256*1024,
    "timeoutMs": 100
  },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
```

```
}  
}
```

Si la solicitud de suscripción se realiza correctamente, la extensión recibe una respuesta correcta HTTP 200.

```
HTTP/1.1 200 OK  
"OK"
```

Si la suscripción falla, la extensión recibe una respuesta de error. Por ejemplo:

```
HTTP/1.1 400 OK  
{  
  "errorType": "ValidationError",  
  "errorMessage": "URI port is not provided; types should not be empty"  
}
```

Estos son algunos códigos de respuesta adicionales que puede recibir la extensión:

- 200: solicitud completada correctamente
- 202: solicitud aceptada. Respuesta a la solicitud de suscripción en un entorno de pruebas locales
- 400: solicitud errónea
- 500: error de servicio

## Referencia del esquema **Event** de la API de telemetría de Lambda

Utilice el punto de conexión de la API de telemetría de Lambda para suscribir extensiones a los flujos de telemetría. Puede recuperar el punto de conexión de la API de telemetría desde la variable de entorno `AWS_LAMBDA_RUNTIME_API`. Para enviar una solicitud de API, agregue la versión de la API (`2022-07-01/`) y `telemetry/`. Por ejemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Para ver la definición de la especificación OpenAPI (OAS) de la versión de respuestas de suscripción `2022-12-13`, consulte lo siguiente:

- HTTP: [telemetry-api-http-schema.zip](#)
- TCP: [telemetry-api-tcp-schema.zip](#)

La siguiente tabla resume de todos los tipos de objetos de Event que admite la API de telemetría.

Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Evento de plataforma	<code>platform.initStart</code>	Se inició la inicialización de la función.	esquema <a href="#">the section called "platform.initStart"</a>
Evento de plataforma	<code>platform.initRuntimeDone</code>	Se completó la inicialización de la función.	esquema <a href="#">the section called "platform.initRuntimeDone"</a>
Evento de plataforma	<code>platform.initReport</code>	Un informe de inicialización de la función.	esquema <a href="#">the section called "platform.initReport"</a>
Evento de plataforma	<code>platform.start</code>	Se inició la invocación de la función.	esquema <a href="#">the section called "platform.start"</a>

Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Evento de plataforma	<code>platform.runtimeDone</code>	El tiempo de ejecución terminó de procesar un evento con éxito o con falla.	esquema <a href="#">the section called “platform.runtimeDone”</a>
Evento de plataforma	<code>platform.report</code>	Un informe de la invocación de funciones.	esquema <a href="#">the section called “platform.report”</a>
Evento de plataforma	<code>platform.restoreStart</code>	Se inició la restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreStart”</a>
Evento de plataforma	<code>platform.restoreRuntimeDone</code>	Se completó la restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreRuntimeDone”</a>
Evento de plataforma	<code>platform.restoreReport</code>	Informe de restauración en tiempo de ejecución.	esquema <a href="#">the section called “platform.restoreReport”</a>
Evento de plataforma	<code>platform.telemetrySubscription</code>	La extensión suscrita a la API de telemetría.	esquema <a href="#">the section called “platform.telemetrySubscription”</a>
Evento de plataforma	<code>platform.logsDropped</code>	Lambda ha eliminado las entradas de registro.	esquema <a href="#">the section called “platform.logsDropped”</a>
Registros de funciones	<code>function</code>	Una línea de registro del código de la función.	esquema <a href="#">the section called “function”</a>



Categoría	Tipo de evento	Descripción	Esquema de registro de eventos
Registros de extensión	extension	Una línea de registro del código de extensión.	esquema <a href="#">the section called “extension”</a>

## Contenido

- [Tipos de objetos Event de la API de telemetría](#)
    - [platform.initStart](#)
    - [platform.initRuntimeDone](#)
    - [platform.initReport](#)
    - [platform.start](#)
    - [platform.runtimeDone](#)
    - [platform.report](#)
    - [platform.restoreStart](#)
    - [platform.restoreRuntimeDone](#)
    - [platform.restoreReport](#)
    - [platform.extension](#)
    - [platform.telemetrySubscription](#)
    - [platform.logsDropped](#)
    - [function](#)
    - [extension](#)
  - [Tipos de objetos compartidos](#)
    - [InitPhase](#)
    - [InitReportMetrics](#)
    - [InitType](#)
    - [ReportMetrics](#)
    - [RestoreReportMetrics](#)
    - [RuntimeDoneMetrics](#)
- Referencia del esquema Event
- [Span](#)

- [Status](#)
- [TraceContext](#)
- [TracingType](#)

## Tipos de objetos **Event** de la API de telemetría

En esta sección, se detallan los tipos de objetos Event que admite la API de telemetría de Lambda. En las descripciones de los eventos, un signo de interrogación (?) indica que el atributo puede no estar presente en el objeto.

### **platform.initStart**

Un evento `platform.initStart` indica que se ha iniciado la fase de inicialización de la función. Un objeto `platform.initStart` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

El objeto `PlatformInitStart` tiene los siguientes atributos:

- `functionName`: String
- `functionVersion`: String
- `InitializationType`: objeto [the section called "InitType"](#)
- `instanceId?`: String
- `instanceMaxMemory?`: Integer
- `phase`: objeto [the section called "InitPhase"](#)
- `runtimeVersion?`: String
- `runtimeVersionArn?`: String

A continuación, se muestra un ejemplo de Event de tipo `platform.initStart`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.initStart",
```

```
"record": {
  "initializationType": "on-demand",
  "phase": "init",
  "runtimeVersion": "nodejs-14.v3",
  "runtimeVersionArn": "arn",
  "functionName": "myFunction",
  "functionVersion": "$LATEST",
  "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
  "instanceMaxMemory": 256
}
```

## platform.initRuntimeDone

Un evento `platform.initRuntimeDone` indica que se ha completado la fase de inicialización de la función. Un objeto `platform.initRuntimeDone Event` tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone
```

El objeto `PlatformInitRuntimeDone` tiene los siguientes atributos:

- `InitializationType`: objeto [the section called "InitType"](#)
- `phase`: objeto [the section called "InitPhase"](#)
- `status`: objeto [the section called "Status"](#)
- `spans?` Lista de objetos [the section called "Span"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.initRuntimeDone`:

```
{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.initRuntimeDone",
  "record": {
    "initializationType": "on-demand"
    "status": "success",
    "spans": [
      {
        "name": "someTimeSpan",
```

```

        "start": "2022-06-02T12:02:33.913Z",
        "durationMs": 70.5
    }
  ]
}

```

## platform.initReport

Un evento `platform.initReport` contiene un informe general de la fase de inicialización de la función. Un objeto `platform.initReport` Event tiene la siguiente forma:

```

Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport

```

El objeto `PlatformInitReport` tiene los siguientes atributos:

- `errorType?`: cadena
- `InitializationType`: objeto [the section called "InitType"](#)
- `phase`: objeto [the section called "InitPhase"](#)
- `metrics`: objeto [the section called "InitReportMetrics"](#)
- `spans?` Lista de objetos [the section called "Span"](#)
- `status`: objeto [the section called "Status"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.initReport`:

```

{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.initReport",
  "record": {
    "initializationType": "on-demand",
    "status": "success",
    "phase": "init",
    "metrics": {
      "durationMs": 125.33
    },
  },
  "spans": [
    {

```

```
        "name": "someTimeSpan",
        "start": "2022-06-02T12:02:33.913Z",
        "durationMs": 90.1
      }
    ]
  }
}
```

## platform.start

Un evento `platform.start` indica que se ha iniciado la fase de invocación de la función. Un objeto `platform.start` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart
```

El objeto `PlatformStart` tiene los siguientes atributos:

- `requestId`: String
- `version?`: String
- `tracing?`: [the section called "TraceContext"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.start`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.start",
  "record": {
    "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
    "version": "$LATEST",
    "tracing": {
      "spanId": "54565fb41ac79632",
      "type": "X-Amzn-Trace-Id",
      "value":
"Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
    }
  }
}
```

## platform.runtimeDone

Un evento `platform.runtimeDone` indica que se ha completado la fase de invocación de la función. Un objeto `platform.runtimeDone Event` tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

El objeto `PlatformRuntimeDone` tiene los siguientes atributos:

- `errorType?: String`
- `metrics?:` objeto [the section called "RuntimeDoneMetrics"](#)
- `requestId: String`
- `status:` objeto [the section called "Status"](#)
- `spans?` Lista de objetos [the section called "Span"](#)
- `tracing?:` objeto [the section called "TraceContext"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.runtimeDone`:

```
{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.runtimeDone",
  "record": {
    "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
    "status": "success",
    "tracing": {
      "spanId": "54565fb41ac79632",
      "type": "X-Amzn-Trace-Id",
      "value":
"Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
    },
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-08-02T12:01:23:521Z",
        "durationMs": 80.0
      }
    ]
  },
}
```

```
    "metrics": {
      "durationMs": 140.0,
      "producedBytes": 16
    }
  }
}
```

## platform.report

Un evento `platform.report` contiene un informe general de la fase de invocación de la función. Un objeto `platform.report` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

El objeto `PlatformReport` tiene los siguientes atributos:

- `metrics`: objeto [the section called "ReportMetrics"](#)
- `requestId`: String
- `spans?` Lista de objetos [the section called "Span"](#)
- `status`: objeto [the section called "Status"](#)
- `tracing?`: objeto [the section called "TraceContext"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.report`:

```
{
  "time": "2022-10-12T00:01:15.000Z",
  "type": "platform.report",
  "record": {
    "metrics": {
      "billedDurationMs": 694,
      "durationMs": 693.92,
      "initDurationMs": 397.68,
      "maxMemoryUsedMB": 84,
      "memorySizeMB": 128
    },
    "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
  }
}
```

```
}
```

## platform.restoreStart

Un evento `platform.restoreStart` indica que se inició un evento de restauración del entorno de funciones. En un evento de restauración del entorno, Lambda crea el entorno a partir de una instantánea almacenada en caché en lugar de inicializarlo desde cero. Para obtener más información, consulte [Lambda SnapStart](#). Un objeto `platform.restoreStart` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

El objeto `PlatformRestoreStart` tiene los siguientes atributos:

- `functionName: String`
- `functionVersion: String`
- `instanceId?: String`
- `instanceMaxMemory?: String`
- `runtimeVersion?: String`
- `runtimeVersionArn?: String`

A continuación, se muestra un ejemplo de Event de tipo `platform.restoreStart`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreStart",
  "record": {
    "runtimeVersion": "nodejs-14.v3",
    "runtimeVersionArn": "arn",
    "functionName": "myFunction",
    "functionVersion": "$LATEST",
    "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
    "instanceMaxMemory": 256
  }
}
```



## `platform.restoreRuntimeDone`

Un evento `platform.restoreRuntimeDone` indica que se completó un evento de restauración del entorno de funciones. En un evento de restauración del entorno, Lambda crea el entorno a partir de una instantánea almacenada en caché en lugar de inicializarlo desde cero. Para obtener más información, consulte [Lambda SnapStart](#). Un objeto `platform.restoreRuntimeDone` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

El objeto `PlatformRestoreRuntimeDone` tiene los siguientes atributos:

- `errorType?: String`
- `spans?` Lista de objetos [the section called "Span"](#)
- `status:` objeto [the section called "Status"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.restoreRuntimeDone`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreRuntimeDone",
  "record": {
    "status": "success",
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-08-02T12:01:23:521Z",
        "durationMs": 80.0
      }
    ]
  }
}
```

## `platform.restoreReport`

Un evento `platform.restoreReport` contiene un informe general de un evento de restauración de funciones. Un objeto `platform.restoreReport` Event tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

El objeto PlatformRestoreReport tiene los siguientes atributos:

- `errorType?`: cadena
- `metrics?`: objeto [the section called "RestoreReportMetrics"](#)
- `spans?` Lista de objetos [the section called "Span"](#)
- `status`: objeto [the section called "Status"](#)

A continuación, se muestra un ejemplo de Event de tipo `platform.restoreReport`:

```
{
  "time": "2022-10-12T00:00:15.064Z",
  "type": "platform.restoreReport",
  "record": {
    "status": "success",
    "metrics": {
      "durationMs": 15.19
    },
    "spans": [
      {
        "name": "someTimeSpan",
        "start": "2022-08-02T12:01:23:521Z",
        "durationMs": 30.0
      }
    ]
  }
}
```

## platform.extension

Un evento `extension` contiene registros del código de extensión. Un objeto `extension Event` tiene la siguiente forma:

```
Event: Object
- time: String
```

```
- type: String = extension
- record: {}
```

El objeto `PlatformExtension` tiene los siguientes atributos:

- `events`: Lista de `String`
- `name`: `String`
- `state`: `String`

A continuación, se muestra un ejemplo de `Event` de tipo `platform.extension`:

```
{
  "time": "2022-10-12T00:02:15.000Z",
  "type": "platform.extension",
  "record": {
    "events": [ "INVOKE", "SHUTDOWN" ],
    "name": "my-telemetry-extension",
    "state": "Ready"
  }
}
```

## **platform.telemetrySubscription**

Un evento `platform.telemetrySubscription` contiene información sobre una suscripción de extensión. Un objeto `platform.telemetrySubscription Event` tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

El objeto `PlatformTelemetrySubscription` tiene los siguientes atributos:

- `name`: `String`
- `state`: `String`
- `types`: lista de `String`

A continuación, se muestra un ejemplo de `Event` de tipo `platform.telemetrySubscription`:

```
{
  "time": "2022-10-12T00:02:35.000Z",
  "type": "platform.telemetrySubscription",
  "record": {
    "name": "my-telemetry-extension",
    "state": "Subscribed",
    "types": [ "platform", "function" ]
  }
}
```

## platform.logsDropped

Un evento `platform.logsDropped` contiene información sobre los eventos eliminados. Lambda emite el evento `platform.logsDropped` cuando una función genera registros a una velocidad demasiado alta para que Lambda los procese. Cuando Lambda no puede enviar registros a CloudWatch o a la extensión suscrita a la API de telemetría a la velocidad a la que su función los produce, elimina los registros para evitar que la ejecución de la función se ralentice. Un objeto `platform.logsDropped Event` tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

El objeto `PlatformLogsDropped` tiene los siguientes atributos:

- `droppedBytes`: Integer
- `droppedRecords`: Integer
- `reason`: String

A continuación, se muestra un ejemplo de Event de tipo `platform.logsDropped`:

```
{
  "time": "2022-10-12T00:02:35.000Z",
  "type": "platform.logsDropped",
  "record": {
    "droppedBytes": 12345,
    "droppedRecords": 123,
    "reason": "Some logs were dropped because the downstream consumer is slower than the logs production rate"
  }
}
```

```
}  
}
```

## function

Un evento `function` contiene registros del código de la función. Un objeto `function Event` tiene la siguiente forma:

```
Event: Object  
- time: String  
- type: String = function  
- record: {}
```

El formato del campo `record` depende de si los registros de la función están formateados en texto sin formato o en formato JSON. Para obtener más información sobre las opciones de configuración del formato de registro, consulte [the section called “Configuración de los formatos de registro JSON y de texto sin formato”](#)

El siguiente es un ejemplo de `Event` de un tipo `function` en el que el formato de registro es texto sin formato:

```
{  
  "time": "2022-10-12T00:03:50.000Z",  
  "type": "function",  
  "record": "[INFO] Hello world, I am a function!"  
}
```

El siguiente es un ejemplo de `Event` de un tipo `function` en el que el formato de registro es texto en formato JSON:

```
{  
  "time": "2022-10-12T00:03:50.000Z",  
  "type": "function",  
  "record": {  
    "timestamp": "2022-10-12T00:03:50.000Z",  
    "level": "INFO",  
    "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",  
    "message": "Hello world, I am a function!"  
  }  
}
```

**Note**

Si la versión del esquema que está utilizando es anterior a la versión 2022-12-13, entonces "record" siempre se representa como una cadena, incluso cuando el formato de registro de la función esté configurado como JSON.

**extension**

Un evento `extension` contiene registros del código de extensión. Un objeto `extension Event` tiene la siguiente forma:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

El formato del campo `record` depende de si los registros de la función están formateados en texto sin formato o en formato JSON. Para obtener más información sobre las opciones de configuración del formato de registro, consulte [the section called "Configuración de los formatos de registro JSON y de texto sin formato"](#)

El siguiente es un ejemplo de `Event` de un tipo `extension` en el que el formato de registro es texto sin formato:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "extension",
  "record": "[INFO] Hello world, I am an extension!"
}
```

El siguiente es un ejemplo de `Event` de un tipo `extension` en el que el formato de registro es texto en formato JSON:

```
{
  "time": "2022-10-12T00:03:50.000Z",
  "type": "extension",
  "record": {
    "timestamp": "2022-10-12T00:03:50.000Z",
    "level": "INFO",
  }
}
```

```
    "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",  
    "message": "Hello world, I am an extension!"  
  }  
}
```

### Note

Si la versión del esquema que está utilizando es anterior a la versión 2022-12-13, entonces "record" siempre se representa como una cadena, incluso cuando el formato de registro de la función esté configurado como JSON.

## Tipos de objetos compartidos

En esta sección, se detallan los tipos de objetos compartidos que admite la API de telemetría de Lambda.

### **InitPhase**

Una enumeración de la cadena que describe la fase en la que se produce el paso de inicialización. En la mayoría de los casos, Lambda ejecuta el código de inicialización de la función durante la fase `init`. Sin embargo, en algunos casos de error, Lambda puede volver a ejecutar el código de inicialización de la función durante la fase `invoke`. (Esto se denomina inicio suprimido).

- Tipo: `String`
- Valores válidos: `init|invoke|snap-start`

### **InitReportMetrics**

Un objeto que contiene métricas sobre una fase de inicialización.

- Tipo: `Object`

Un objeto `InitReportMetrics` tiene la siguiente forma:

```
InitReportMetrics: Object  
- durationMs: Double
```

A continuación, se muestra un ejemplo de objeto `InitReportMetrics`:

```
{
  "durationMs": 247.88
}
```

## InitType

Una enumeración de la cadena que describe cómo Lambda inicializó el entorno.

- Tipo: String
- Valores válidos: on-demand|provisioned-concurrency

## ReportMetrics

Un objeto que contiene métricas sobre una fase completa.

- Tipo: Object

Un objeto ReportMetrics tiene la siguiente forma:

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

A continuación, se muestra un ejemplo de objeto ReportMetrics:

```
{
  "billedDurationMs": 694,
  "durationMs": 693.92,
  "initDurationMs": 397.68,
  "maxMemoryUsedMB": 84,
  "memorySizeMB": 128
}
```

## RestoreReportMetrics

Un objeto que contiene métricas sobre una fase de restauración completa.



- Tipo: Object

Un objeto `RestoreReportMetrics` tiene la siguiente forma:

```
RestoreReportMetrics: Object
- durationMs: Double
```

A continuación, se muestra un ejemplo de objeto `RestoreReportMetrics`:

```
{
  "durationMs": 15.19
}
```

## RuntimeDoneMetrics

Un objeto que contiene métricas sobre una fase de invocación completa.

- Tipo: Object

Un objeto `RuntimeDoneMetrics` tiene la siguiente forma:

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

A continuación, se muestra un ejemplo de objeto `RuntimeDoneMetrics`:

```
{
  "durationMs": 200.0,
  "producedBytes": 15
}
```

## Span

Un objeto que contiene detalles acerca de un intervalo. Un intervalo representa una unidad de trabajo u operación en un seguimiento. Para obtener más información sobre intervalos, consulte [Span](#) en la página de la API de seguimiento del sitio web de OpenTelemetry Docs.

Lambda admite los siguientes intervalos para el evento `platform.RuntimeDone`:

- El intervalo `responseLatency` describe cuánto tiempo tardó la función de Lambda en empezar a enviar la respuesta.
- El intervalo `responseDuration` describe cuánto tiempo tardó la función de Lambda en terminar de enviar toda la respuesta.
- El intervalo `runtimeOverhead` describe cuánto tiempo tardó el tiempo de ejecución de Lambda en indicar que estaba listo para procesar la siguiente invocación de la función. Este es el tiempo que tardó el tiempo de ejecución en llamar a la API de [siguiente invocación](#) para obtener el siguiente evento después de devolver la respuesta de su función.

A continuación, se muestra un ejemplo de objeto de intervalo `responseLatency`:

```
{
  "name": "responseLatency",
  "start": "2022-08-02T12:01:23.521Z",
  "durationMs": 23.02
}
```

## Status

Un objeto que describe el estado de una fase de inicialización o invocación. Si el estado es `failure` o `error`, el objeto `Status` también contiene un campo `errorType` que describe el error.

- Tipo: `Object`
- Valores de estado válidos: `success|failure|error|timeout`

## TraceContext

Un objeto que describe las propiedades de un seguimiento.

- Tipo: `Object`

Un objeto `TraceContext` tiene la siguiente forma:

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

A continuación, se muestra un ejemplo de objeto TraceContext:

```
{
  "spanId": "073a49012f3c312e",
  "type": "X-Amzn-Trace-Id",
  "value":
  "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

## TracingType

Una enumeración de la cadena que describe el tipo de seguimiento de un objeto [the section called "TraceContext"](#).

- Tipo: String
- Valores válidos: X-Amzn-Trace-Id

## Conversión de objetos **Event** de la API de telemetría de Lambda en OpenTelemetry Spans

El esquema de la API de telemetría AWS Lambda es semánticamente compatible con OpenTelemetry (OTel). Esto significa que puede convertir sus objetos Event de la API de telemetría AWS Lambda en OpenTelemetry (OTel) Spans. Al realizar la conversión, no debe asignar un solo objeto de Event a un solo OTel Span. En su lugar, debe presentar los tres eventos relacionados con una fase del ciclo de vida en un solo OTel Span. Por ejemplo, los eventos `start`, `runtimeDone` y `runtimeReport` representan una sola invocación de función. Presente estos tres eventos como un solo OTel Span.

Puede convertir sus eventos mediante Span Events o Child Spans (agrupados). Las tablas de esta página describen las asignaciones entre las propiedades del esquema de la API de telemetría y las propiedades de OTel Span para ambos enfoques. Para obtener más información sobre OTel Spans, consulte [Span](#) en la página de la API de seguimiento del sitio web de OpenTelemetry Docs.

### Secciones

- [Mapeo de OTel Spans con Span Events](#)
- [Mapeo de OTel Spans con Child Spans](#)

### Mapeo de OTel Spans con Span Events

En las siguientes tablas, `e` representa el evento que proviene del origen de telemetría.

#### Mapeo de los eventos \*Start

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.Name</code>	Su extensión genera este valor en función del campo <code>type</code> .
<code>Span.StartTime</code>	Utilice <code>e.time</code> .
<code>Span.EndTime</code>	N/D, porque el evento aún no se ha completado.
<code>Span.Kind</code>	Configurado en <code>Server</code> .

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.Status</code>	Configurado en <code>Unset</code> .
<code>Span.TraceId</code>	Analice el encabezado AWS X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>TraceId</code> .
<code>Span.ParentId</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Parent</code> .
<code>Span.SpanId</code>	Utilice <code>e.tracing.spanId</code> si está disponible. De lo contrario, genere un nuevo <code>SpanId</code> .
<code>Span.SpanContext.TraceState</code>	N/D para un contexto de seguimiento de X-Ray.
<code>Span.SpanContext.TraceFlags</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Sampled</code> .
<code>Span.Attributes</code>	Su extensión puede agregar aquí cualquier valor personalizado.

#### Mapeo de los eventos `*RuntimeDone`

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.Name</code>	Su extensión genera el valor en función del campo <code>type</code> .
<code>Span.StartTime</code>	Utilice <code>e.time</code> desde el evento coincidente <code>*Start</code> .  Para otras opciones, consulte <code>e.time - e.metrics.durationMs</code> .

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.EndTime</code>	N/D, porque el evento aún no se ha completado.
<code>Span.Kind</code>	Configurado en <code>Server</code> .
<code>Span.Status</code>	<p>Si <code>e.status</code> no es igual a <code>success</code>, entonces configúrelo en <code>Error</code>.</p> <p>De lo contrario, establézcala en <code>Ok</code>.</p>
<code>Span.Events[]</code>	Utilice <code>e.spans[]</code> .
<code>Span.Events[i].Name</code>	Utilice <code>e.spans[i].name</code> .
<code>Span.Events[i].Time</code>	Utilice <code>e.spans[i].start</code> .
<code>Span.TraceId</code>	Analice el encabezado AWS X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>TraceId</code> .
<code>Span.ParentId</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Parent</code> .
<code>Span.SpanId</code>	Use el mismo <code>SpanId</code> del evento <code>*Start</code> . Si no está disponible, entonces utilice <code>e.tracing.spanId</code> o genere un nuevo <code>SpanId</code> .
<code>Span.SpanContext.TraceState</code>	N/D para un contexto de seguimiento de X-Ray.
<code>Span.SpanContext.TraceFlags</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Sampled</code> .
<code>Span.Attributes</code>	Su extensión puede agregar aquí cualquier valor personalizado.

## Mapeo de los eventos \*Report

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.Name</code>	Su extensión genera el valor en función del campo <code>type</code> .
<code>Span.StartTime</code>	Utilice <code>e.time</code> desde el evento coincidente <code>*Start</code> .  Para otras opciones, consulte <code>e.time - e.metrics.durationMs</code> .
<code>Span.EndTime</code>	Utilice <code>e.time</code> .
<code>Span.Kind</code>	Configurado en <code>Server</code> .
<code>Span.Status</code>	Usa el mismo valor que el evento <code>*RuntimeDone</code> .
<code>Span.TraceId</code>	Analice el encabezado AWS X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>TraceId</code> .
<code>Span.ParentId</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Parent</code> .
<code>Span.SpanId</code>	Use el mismo <code>SpanId</code> del evento <code>*Start</code> . Si no está disponible, entonces utilice <code>e.tracing.spanId</code> o genere un nuevo <code>SpanId</code> .
<code>Span.SpanContext.TraceState</code>	N/D para un contexto de seguimiento de X-Ray.
<code>Span.SpanContext.TraceFlags</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Sampled</code> .

OpenTelemetry	Esquema de la API de telemetría de Lambda
Span.Attributes	Su extensión puede agregar aquí cualquier valor personalizado.

## Mapeo de OTel Spans con Child Spans

En la siguiente tabla se describe cómo convertir los eventos de la API de telemetría de Lambda en OTel Spans con Child Spans (agrupados) para los Spans `*RuntimeDone`. Para las asignaciones de `*Start` y `*Report`, consulte las tablas en [the section called “Mapeo de OTel Spans con Span Events”](#), ya que son las mismas para Child Spans. En esta tabla, `e` representa el evento que proviene del origen de telemetría.

### Mapeo de los eventos `*RuntimeDone`

OpenTelemetry	Esquema de la API de telemetría de Lambda
Span.Name	Su extensión genera el valor en función del campo <code>type</code> .
Span.StartTime	Utilice <code>e.time</code> desde el evento coincidente <code>*Start</code> .  Para otras opciones, consulte <code>e.time - e.metrics.durationMs</code> .
Span.EndTime	N/D, porque el evento aún no se ha completado.
Span.Kind	Configurado en <code>Server</code> .
Span.Status	Si <code>e.status</code> no es igual a <code>success</code> , entonces configúrelo en <code>Error</code> .  De lo contrario, establézcala en <code>Ok</code> .



OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>Span.TraceId</code>	Analice el encabezado AWS X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>TraceId</code> .
<code>Span.ParentId</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Parent</code> .
<code>Span.SpanId</code>	Use el mismo <code>SpanId</code> del evento <code>*Start</code> . Si no está disponible, entonces utilice <code>e.tracing.spanId</code> o genere un nuevo <code>SpanId</code> .
<code>Span.SpanContext.TraceState</code>	N/D para un contexto de seguimiento de X-Ray.
<code>Span.SpanContext.TraceFlags</code>	Analice el encabezado X-Ray que se encuentra en <code>e.tracing.value</code> y, a continuación, use el valor <code>Sampled</code> .
<code>Span.Attributes</code>	Su extensión puede agregar aquí cualquier valor personalizado.
<code>ChildSpan[i].Name</code>	Utilice <code>e.spans[i].name</code> .
<code>ChildSpan[i].StartTime</code>	Utilice <code>e.spans[i].start</code> .
<code>ChildSpan[i].EndTime</code>	Utilice <code>e.spans[i].start + e.spans[i].durations</code> .
<code>ChildSpan[i].Kind</code>	Igual que el <code>Span.Kind</code> principal.
<code>ChildSpan[i].Status</code>	Igual que el <code>Span.Status</code> principal.
<code>ChildSpan[i].TraceId</code>	Igual que el <code>Span.TraceId</code> principal.
<code>ChildSpan[i].ParentId</code>	Utilice el <code>Span.SpanId</code> principal.
<code>ChildSpan[i].SpanId</code>	Genere una nueva <code>SpanId</code> .

OpenTelemetry	Esquema de la API de telemetría de Lambda
<code>ChildSpan[i].SpanContext.TraceState</code>	N/D para un contexto de seguimiento de X-Ray.
<code>ChildSpan[i].SpanContext.TraceFlags</code>	Igual que el <code>Span.SpanContext.TraceFlags</code> principal.

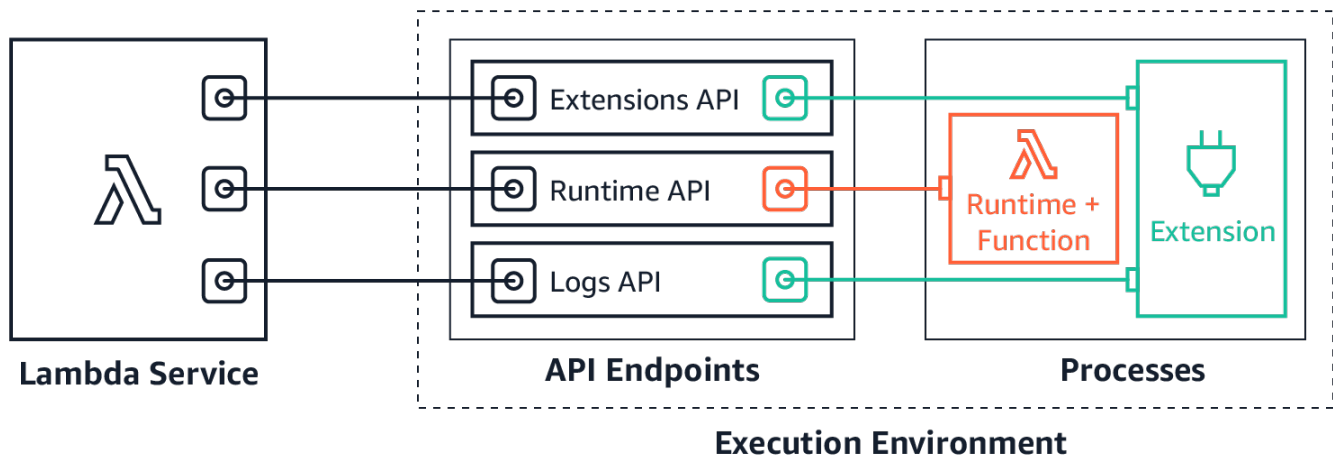
## Uso de la API de registros de Lambda

### ⚠ Important

La API de telemetría de Lambda reemplaza a la API de registros de Lambda. Si bien la API de registros sigue siendo completamente funcional, recomendamos utilizar solo la API de telemetría en el futuro. Puede suscribirse su extensión a una transmisión de telemetría mediante la API de telemetría o la API de registros. Tras suscribirse mediante una de estas API, cualquier intento de suscribirse mediante la otra API volverá a causar un error.


Lambda captura automáticamente los registros de tiempo de ejecución y los transmite a Amazon CloudWatch. Esta secuencia de registro contiene los registros que generan el código de función y las extensiones, así como los registros que genera Lambda como parte de la invocación de la función.

[Las extensiones Lambda](#) pueden usar la API de registros de tiempo de ejecución de Lambda para suscribirse a flujos de registro directamente desde el [entorno de ejecución](#). Lambda transfiere los registros a la extensión y la extensión puede procesar, filtrar y enviar los registros a cualquier destino preferido.



La API Logs permite que las extensiones se suscriban a tres flujos de registros diferentes:

- Registros de funciones que la función de Lambda genera y escribe en `stdout` o `stderr`.
- La extensión registra que genera el código de extensión.
- Registros de plataforma de Lambda en los que se registran eventos y errores relacionados con invocaciones y extensiones.

 Note

Lambda envía todos los registros a CloudWatch, incluso cuando una extensión se suscribe a una o más de los flujos de registro.


## Temas

- [Suscribirse para recibir registros](#)
- [Uso de memoria](#)
- [Protocolos de destino](#)
- [Configuración de almacenamiento en búfer](#)
- [Ejemplo de suscripción](#)
- [Código de ejemplo para Logs API](#)
- [Referencia de la API de registros](#)
- [Mensajes de registro](#)

## Suscribirse para recibir registros

Una extensión de Lambda puede suscribirse para recibir registros enviando una solicitud de suscripción a la API de registros.

Para suscribirse para recibir registros, necesita el identificador de extensión (Lambda-Extension-Identifier). Primero [registre la extensión](#) para recibir el identificador de extensión. A continuación, suscríbase a la API de registros durante [la inicialización](#). Una vez finalizada la fase de inicialización, Lambda no procesa las solicitudes de suscripción.

 Note

La suscripción a la API de registros es idempotente. Las solicitudes de suscripción duplicadas no dan lugar a suscripciones duplicadas.

## Uso de memoria

El uso de memoria aumenta linealmente a medida que aumenta el número de suscriptores. Las suscripciones consumen recursos de memoria porque cada suscripción abre un nuevo búfer de

memoria para almacenar los registros. Para ayudar a optimizar el uso de memoria, puede ajustar la [configuración de almacenamiento en búfer](#). El uso de memoria del búfer cuenta para el consumo general de memoria en el entorno de ejecución.

## Protocolos de destino

Puede elegir uno de los siguientes protocolos para recibir los registros:

1. HTTP (recomendado): Lambda entrega registros a un punto de enlace HTTP local (`http://sandbox.localdomain:${PORT}/${PATH}`) como una matriz de registros en formato JSON. El parámetro `$PATH` es opcional. Tenga en cuenta que solo se admite HTTP, no HTTPS. Puede optar por recibir registros a través de PUT o POST.
2. TCP: Lambda entrega registros a un puerto TCP en [formato JSON delimitado por línea nueva \(NDJSON\)](#).

Recomendamos utilizar HTTP en lugar de TCP. Con TCP, la plataforma de Lambda no puede reconocer cuando los registros se entregan a la capa de aplicación. Por lo tanto, puede perder registros si su extensión se bloquea. HTTP no comparte esta limitación.

También recomendamos configurar el agente de escucha HTTP local o el puerto TCP antes de suscribirse para recibir registros. Durante la instalación, tenga en cuenta lo siguiente:

- Lambda envía registros solo a destinos que se encuentran dentro del entorno de ejecución.
- Lambda intenta volver a enviar los registros (con retroceso) si no hay ningún agente de escucha o si la solicitud POST o PUT da como resultado un error. Si el suscriptor del registro se bloquea, continuará recibiendo registros después de que Lambda reinicie el entorno de ejecución.
- Lambda reserva el puerto 9001. No hay otras restricciones ni recomendaciones sobre el número de puerto.

## Configuración de almacenamiento en búfer

Lambda puede almacenar en búfer los registros y entregarlos al suscriptor. Puede configurar este comportamiento en la solicitud de suscripción al especificar los siguientes campos opcionales: Tenga en cuenta que Lambda utiliza el valor predeterminado para cualquier campo que no especifique.

- `timeoutMs`: el tiempo máximo (en milisegundos) para almacenar en búfer un lote.  
Predeterminado: 1000. Mínimo: 25. Máximo: 30 000.

- **maxBytes**: el tamaño máximo (en bytes) de los registros que se deben almacenar en memoria. Predeterminado: 262 144. Mínimo: 262 144 Máximo: 1 048 576
- **maxItems**: el número máximo de eventos que se deben almacenar en memoria. Predeterminado: 10 000. Mínimo: 1000. Máximo: 10 000.

Durante la configuración del almacenamiento en búfer, tenga en cuenta los siguientes puntos:

- Lambda vacía los registros si alguno de los flujos de entrada está cerrado, por ejemplo, si el tiempo de ejecución se bloquea.
- Cada suscriptor puede especificar una configuración de almacenamiento en búfer diferente en la solicitud de suscripción.
- Considere el tamaño del búfer que necesita para leer los datos. Espere recibir cargas útiles tan grandes como  $2 * \text{maxBytes} + \text{metadata}$ , donde **maxBytes** está configurado en la solicitud de suscripción. Por ejemplo, Lambda agrega los siguientes bytes de metadatos a cada registro:

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "function",
  "record": "Hello World"
}
```

- Si el suscriptor no puede procesar los registros entrantes con la suficiente rapidez, Lambda podría soltar registros para mantener limitada la utilización de la memoria. Para indicar el número de registros eliminados, Lambda envía un registro `platform.logsDropped`. Para obtener más información, consulte [the section called “Lambda: no aparecen todos los registros de mi función”](#).

## Ejemplo de suscripción

En el siguiente ejemplo se muestra una solicitud para suscribirse a los registros de la plataforma y de las funciones.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs HTTP/1.1
{ "schemaVersion": "2020-08-15",
  "types": [
    "platform",
    "function"
  ],
  "buffering": {
```

```
    "maxItems": 1000,
    "maxBytes": 262144,
    "timeoutMs": 100
  },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080/lambda_logs"
  }
}
```

Si la solicitud se realiza correctamente, el suscriptor recibe una respuesta correcta HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

## Código de ejemplo para Logs API

Para obtener un código de ejemplo que muestra cómo enviar registros a un destino personalizado, consulte [Uso de extensiones de AWS Lambda para enviar registros a destinos personalizados](#) en el blog de informática de AWS.

Para ver ejemplos de código de Python and Go que muestran cómo desarrollar una extensión básica de Lambda y suscribirse a la API de registros, consulte [Extensiones de AWS Lambda](#) en el repositorio de ejemplos de AWS de GitHub. Para obtener más información sobre cómo crear una extensión de Lambda, consulte [the section called “API de extensiones”](#).

## Referencia de la API de registros

Puede recuperar el extremo de la API de registros desde la variable de entorno `AWS_LAMBDA_RUNTIME_API`. Para enviar una solicitud de API, utilice el prefijo `2020-08-15/` antes de la ruta de la API. Por ejemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

La especificación OpenAPI para la solicitud de suscripción de registros de la API, versión 2020-08-15, está disponible aquí: [logs-api-request.zip](#)

## Suscribirse

Para suscribirse a una o más de los flujos de registro que están disponibles en el entorno de ejecución de Lambda, las extensiones envían una solicitud de Suscribe API.

## Ruta – /logs

Método: PUT

Body parameters (Parámetros del cuerpo)

`destination`: consulte [the section called “Protocolos de destino”](#). Requerido: sí. Tipo: cadenas.

`buffering`: consulte [the section called “Configuración de almacenamiento en búfer”](#). Obligatorio: no  
Tipo: cadenas.

`types`: matriz de los tipos de registros que se deben recibir. Requerido: sí. Tipo: matriz de cadenas.  
Valores válidos: «plataforma», «función», «extensión».

`schemaVersion`: Obligatorio: no. Valor predeterminado: “2020-08-15”. Establezca el valor en “2021-03-18” para que la extensión reciba mensajes [platform.runtimeDone](#).

### Parámetros de respuesta

Las especificaciones de OpenAPI para las respuestas de suscripción, versión 2020-08-15, están disponibles para protocolos HTTP y TCP:

- HTTP: [logs-api-http-response.zip](#)
- TCP: [logs-api-tcp-response.zip](#)

### Códigos de respuesta

- 200: solicitud completada correctamente
- 202: solicitud aceptada. Respuesta a una solicitud de suscripción durante las pruebas locales.
- 4XX: solicitud errónea
- 500: error de servicio

Si la solicitud se realiza correctamente, el suscriptor recibe una respuesta correcta HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Si la solicitud falla, el suscriptor recibe una respuesta de error. Por ejemplo:



```
HTTP/1.1 400 OK
{
  "errorType": "Logs.ValidationError",
  "errorMessage": "URI port is not provided; types should not be empty"
}
```

## Mensajes de registro

La API Logs permite que las extensiones se suscriban a tres flujos de registros diferentes:

- **Función:** registros que la función de Lambda genera y escribe en `stdout` o `stderr`.
- **Extensión:** los registros de extensión generados por el código de extensión.
- **Plataforma:** registros de la plataforma generados por la plataforma de tiempo de ejecución, en los que se registran eventos y errores relacionados con invocaciones y extensiones.

## Temas

- [Registros de funciones](#)
- [Registros de extensión](#)
- [Registros de plataforma](#)

## Registros de funciones

La función de Lambda y las extensiones internas generan registros de funciones y los escriben en `stdout` o `stderr`.

En el ejemplo siguiente, se muestra el formato de un mensaje de registro de función. `{ "time": "2020-08-20T12:31:32.123Z", "type": "function", "record": "ERROR encountered. Stack trace:\n\nmy-function (line 10)\n" }`

## Registros de extensión

Las extensiones pueden generar registros de extensión. El formato de registro es el mismo que para un registro de función.

## Registros de plataforma

Lambda genera mensajes de registro para eventos de plataforma como `platform.start`, `platform.end` y `platform.fault`.

Opcionalmente, puede suscribirse a la versión 2021-03-18 del esquema de la API de registros, que incluye el mensaje de registro `platform.runtimeDone`.

Ejemplo de mensajes de registros de plataforma:

En el siguiente ejemplo, se muestran los registros de inicio y final de registro de la plataforma. Estos registros indican la hora de inicio de la invocación y la hora de finalización de la invocación que especifica el `requestId`.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.start",
  "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.end",
  "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
```

El mensaje de registro `Platform.IniRuntimeDone` muestra el estado de la subfase `Runtime init`, que forma parte de [la fase del ciclo de vida de Init](#). Cuando `Runtime init` tiene éxito, el tiempo de ejecución envía una solicitud de API `/next` en tiempo de ejecución (para los tipos de inicialización `on-demand` y `provisioned-concurrency`) o `restore/next` (para el tipo de inicialización `snap-start`). El siguiente ejemplo muestra un mensaje de registro de `Platform.IniRuntimeDone` correcto para el tipo de inicialización `snap-start`.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initRuntimeDone",
  "record":{"
    "initializationType":"snap-start",
    "status":"success"
  }}
}
```

El mensaje de registro `Platform.initReport` muestra cuánto duró la fase `Init` y cuántos milisegundos se le facturaron durante esta fase. Cuando el tipo de inicialización es `provisioned-concurrency`, Lambda envía este mensaje durante la invocación. Cuando el tipo de inicialización es `snap-start`, Lambda envía este mensaje después de restaurar la imagen instantánea. El siguiente ejemplo

muestra un mensaje de registro de Platform.IniRuntimeDone para el tipo de inicialización snap-start.

```
{
  "time": "2022-07-17T18:41:57.083Z",
  "type": "platform.initReport",
  "record": {
    "initializationType": "snap-start",
    "metrics": {
      "durationMs": 731.79,
      "billedDurationMs": 732
    }
  }
}
```

El registro de informes de plataforma incluye métricas sobre la invocación que especifica el requestId. El campo initDurationMs se incluye en el registro solamente si la invocación incluye un inicio en frío. Si el seguimiento AWS X-Ray está activo, el registro incluye metadatos de X-Ray. En el siguiente ejemplo se muestra un registro de informes de plataforma para una invocación que incluyó un inicio en frío.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.report",
  "record": { "requestId": "6f7f0961f83442118a7af6fe80b88d56",
    "metrics": { "durationMs": 101.51,
      "billedDurationMs": 300,
      "memorySizeMB": 512,
      "maxMemoryUsedMB": 33,
      "initDurationMs": 116.67
    }
  }
}
```

El registro de la plataforma captura errores de tiempo de ejecución o del entorno de ejecución. En el siguiente ejemplo se muestra un mensaje de registro de errores de la plataforma.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.fault",
```

```
"record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
completing request"
}
```

### Note

AWS está implementando cambios en el servicio Lambda. Debido a estos cambios, es posible que vea pequeñas diferencias entre la estructura y el contenido de los mensajes de registro del sistema y los segmentos de rastro emitidos por diferentes funciones de Lambda en su Cuenta de AWS.

Uno de los resultados de registro afectados por este cambio es el campo "record" de registro de errores de la plataforma. Los siguientes ejemplos muestran campos "record" ilustrativos en los formatos antiguo y nuevo. El nuevo estilo de registro de errores contiene un mensaje más conciso

Estos cambios se implementarán en las próximas semanas y todas las funciones de todas las Regiones de AWS, excepto en las regiones de China y GovCloud, pasarán a utilizar el nuevo formato de mensajes de registro y segmentos de rastro.

### Example registro de errores de la plataforma (estilo antiguo)

```
"record": "RequestId: ... \tError: Runtime exited with error: exit status
255\nRuntime.ExitError"
```

### Example registro de errores de la plataforma (estilo nuevo)

```
"record": "RequestId: ... Status: error \tErrorType: Runtime.ExitError"
```

Lambda genera un registro de extensión de plataforma cuando una extensión se registra con la API de extensiones. En el siguiente ejemplo, se muestra un mensaje de registro de errores de la plataforma.

```
{
  "time": "2020-08-20T12:31:32.123Z",
  "type": "platform.extension",
  "record": {"name": "Foo.bar",
    "state": "Ready",
    "events": ["INVOKE", "SHUTDOWN"]
  }
}
```

```
}  
}
```

Lambda genera un registro de suscripción de registros de plataforma cuando una extensión se suscribe a la API de registros. En el ejemplo siguiente, se muestra un mensaje de registro de suscripción.

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "platform.logsSubscription",  
  "record": {"name": "Foo.bar",  
    "state": "Subscribed",  
    "types": ["function", "platform"]},  
}
```

Lambda genera un registro de registros eliminados de la plataforma cuando una extensión no puede procesar el número de registros que recibe. En el siguiente ejemplo se muestra un mensaje de registro `platform.logsDropped`.

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "platform.logsDropped",  
  "record": {"reason": "Consumer seems to have fallen behind as it has not  
acknowledged receipt of logs.",  
    "droppedRecords": 123,  
    "droppedBytes": 12345},  
}
```

El mensaje de registro `Platform.RestoreRestart` muestra la hora en que se inició la fase `Restore` (solo el tipo de inicialización `snap-start`). Ejemplo:

```
{  
  "time": "2022-07-17T18:43:44.782Z",  
  "type": "platform.restoreStart",  
  "record": {}  
}
```

El mensaje de registro `Platform.initReport` muestra cuánto duró la fase `Restore` y cuántos milisegundos se le facturaron durante esta fase (solo el tipo de inicialización `snap-start`). Ejemplo:

```
{
  "time": "2022-07-17T18:43:45.936Z",
  "type": "platform.restoreReport",
  "record": {
    "metrics": {
      "durationMs": 70.87,
      "billedDurationMs": 13
    }
  }
}
```

## Mensajes `runtimeDone` de la plataforma

Si establece la versión del esquema en "2021-03-18" en la solicitud de suscripción, Lambda envía un mensaje `platform.runtimeDone` después de que la invocación de la función se complete correctamente o presente un error. La extensión puede utilizar este mensaje para detener toda la recopilación de telemetría para esta invocación de función.

La especificación OpenAPI para el tipo de evento de registro en la versión de esquema 2021-03-18 está disponible aquí: [schema-2021-03-18.zip](#)

Lambda genera el mensaje de registro `platform.runtimeDone` cuando el tiempo de ejecución envía una solicitud `Next` o `Error` de la API de tiempo de ejecución. El registro `platform.runtimeDone` informa a los consumidores de registros de la API que la invocación de función se completa. Las extensiones pueden utilizar esta información para decidir cuándo enviar toda la telemetría recopilada durante esa invocación.

## Ejemplos

Lambda envía el mensaje `platform.runtimeDone` después de que el tiempo de ejecución envía la solicitud `NEXT` cuando se completa la invocación de la función. Los ejemplos siguientes muestran mensajes para cada uno de los valores de estado: éxito, error y tiempo de espera agotado.

### Example Ejemplo de mensaje de éxito

```
{
  "time": "2021-02-04T20:00:05.123Z",
  "type": "platform.runtimeDone",
  "record": {
    "requestId": "6f7f0961f83442118a7af6fe80b88",
    "status": "success"
  }
}
```

```
}  
}
```

### Example Ejemplo de mensaje de error

```
{  
  "time": "2021-02-04T20:00:05.123Z",  
  "type": "platform.runtimeDone",  
  "record": {  
    "requestId": "6f7f0961f83442118a7af6fe80b88",  
    "status": "failure"  
  }  
}
```

### Example Ejemplo de mensaje de tiempo de espera agotado

```
{  
  "time": "2021-02-04T20:00:05.123Z",  
  "type": "platform.runtimeDone",  
  "record": {  
    "requestId": "6f7f0961f83442118a7af6fe80b88",  
    "status": "timeout"  
  }  
}
```

### Example Ejemplo de mensaje Platform.RestoreUntimeDone (solo tipo de inicialización **snap-start**)

El mensaje de registro Platform.RestoreUntimeDone muestra si la fase se realizó Restore correctamente o no. Lambda envía este mensaje cuando el tiempo de ejecución envía una solicitud de API `restore/next` de tiempo de ejecución. Hay tres estados posibles: éxito, error y tiempo de espera agotado. En el ejemplo siguiente se muestra un mensaje de registro de Platform.RestoreRuntimeDone correcto.

```
{  
  "time": "2022-07-17T18:43:45.936Z",  
  "type": "platform.restoreRuntimeDone",  
  "record": {  
    "status": "success"  
  }  
}
```

# Solución de problemas de Lambda

Los temas siguientes proporcionan consejos para solucionar problemas de errores y problemas que puedan surgir al utilizar la API de Lambda, la consola o las herramientas. Si se encuentra con un problema que no aparezca en esta lista, puede utilizar el botón Comentarios de esta página para notificarlo.

Para obtener más consejos sobre la resolución de problemas y respuestas a preguntas comunes de soporte, visite el [Centro de conocimientos de AWS](#).

Para obtener más información acerca de la depuración y la resolución de problemas de las aplicaciones de Lambda, consulte [Debugging](#) en Serverless Land.

## Temas

- [Solución de problemas en las implementaciones de Lambda](#)
- [Solucionar problemas de invocación en Lambda](#)
- [Solucionar problemas de ejecución en Lambda](#)
- [Solucionar problemas de redes en Lambda](#)

## Solución de problemas en las implementaciones de Lambda

Al actualizar su función, Lambda implementa el cambio iniciando nuevas instancias de la función con el código o la configuración actualizados. Los errores de implementación impiden que se utilice la nueva versión y pueden deberse a problemas con el paquete de implementación, el código, los permisos o las herramientas.

Cuando implementa actualizaciones en su función directamente con la API de Lambda o con un cliente como la AWS CLI, puede ver errores de Lambda directamente en la salida. Si utiliza servicios como AWS CloudFormation, AWS CodeDeploy o AWS CodePipeline, busque la respuesta de Lambda en los registros o en la secuencia de eventos del servicio.

Los temas siguientes proporcionan consejos para solucionar problemas de errores y problemas que puedan surgir al utilizar la API de Lambda, la consola o las herramientas. Si se encuentra con un problema que no aparezca en esta lista, puede utilizar el botón Comentarios de esta página para notificarlo.



Para obtener más consejos sobre la resolución de problemas y respuestas a preguntas comunes de soporte, visite el [Centro de conocimientos de AWS](#).

Para obtener más información acerca de la depuración y la resolución de problemas de las aplicaciones de Lambda, consulte [Debugging](#) en Serverless Land.

## Temas

- [General: Se deniega el permiso/No se puede cargar dicho archivo](#)
- [General: se produce un error al llamar al UpdateFunctionCode](#)
- [Amazon S3: Código de error PermanentRedirect.](#)
- [General: No se puede encontrar, no se puede cargar, no se puede importar, clase no encontrada, ningún archivo o directorio](#)
- [General: controlador de método no definido](#)
- [Lambda: error de conversión de la capa](#)
- [Lambda: InvalidParameterValueException o RequestEntityTooLargeException](#)
- [Lambda: InvalidParameterValueException](#)
- [Lambda: cuotas de simultaneidad y memoria](#)

## General: Se deniega el permiso/No se puede cargar dicho archivo

Error: EACCES: permission denied, open '/var/task/index.js'

Error: cannot load such file -- function

Error: [Errno 13] Permission denied: '/var/task/function.py'

El tiempo de ejecución de Lambda necesita permiso para leer los archivos del paquete de implementación. En la notación octal de permisos de Linux, Lambda necesita 644 permisos para archivos no ejecutables (rw-r--r--) y 755 permisos (rwxr-xr-x) para directorios y archivos ejecutables.

En Linux y macOS, utilice el comando `chmod` para cambiar los permisos de los archivos y directorios del paquete de implementación. Por ejemplo, para brindarle a un archivo ejecutable los permisos correctos, ejecute el siguiente comando.

```
chmod 755 <filepath>
```

Para cambiar los permisos de los archivos en Windows, consulte [Set, View, Change, or Remove Permissions on an Object](#) en la documentación de Microsoft Windows.

## General: se produce un error al llamar al UpdateFunctionCode

Error: An error occurred (RequestEntityTooLargeException) when calling the UpdateFunctionCode operation

Cuando carga un paquete de implementación o un archivo de capa directamente en Lambda, el tamaño del archivo ZIP está limitado a 50 MB. Para cargar un archivo más grande, almacénelo en Amazon S3 y utilice los parámetros S3Bucket y S3Key.

### Note

Cuando carga un archivo directamente con la AWS CLI, el SDK de AWS o de otro modo, el archivo ZIP binario se convierte a base64, que aumenta su tamaño en aproximadamente un 30 %. Para permitir esto y el tamaño de otros parámetros de la solicitud, el límite de tamaño de solicitud real que aplica Lambda es mayor. Por este motivo, el límite de 50 MB es aproximado.

## Amazon S3: Código de error PermanentRedirect.

Error: se produjo un error mientras GetObject. Código de error S3: PermanentRedirect. Mensaje de error S3: el bucket está en esta región: us-east-2. Utilice esta región para volver a intentar la solicitud

Cuando carga el paquete de implementación de una función desde un bucket Amazon S3, el bucket debe estar en la misma Región que la función. Este problema puede producirse cuando especifica un objeto Amazon S3 en una llamada a [UpdateFunctionCode](#), o utiliza el paquete y los comandos de implementación en la CLI de AWS CLI o en la CLI de AWS SAM. Cree un bucket de artefactos de implementación para cada Región donde desarrolle aplicaciones.

General: No se puede encontrar, no se puede cargar, no se puede importar, clase no encontrada, ningún archivo o directorio

Error: Cannot find module 'function'

Error: cannot load such file -- function

Error: Unable to import module 'function'

Error: Class not found: function.Handler

Error: fork/exec /var/task/function: no such file or directory

Error: Unable to load type 'Function.Handler' from assembly 'Function'.

El nombre del archivo o clase en la configuración del controlador de su función no coincide con su código. Consulte la siguiente sección para obtener más información.

## General: controlador de método no definido

Error: index.handler is undefined or not exported

Error: Handler 'handler' missing on module 'function'

Error: undefined method `handler' for #<LambdaHandler:0x000055b76cceb98>

Error: No public method named handleRequest with appropriate method signature found on class function.Handler

Error: Unable to find method 'handleRequest' in type 'Function.Handler' from assembly 'Function'

El nombre del método de controlador en la configuración del controlador de su función no coincide con su código. Cada motor de ejecución define una convención de nomenclatura para los controladores, como *nombredearchivo.nombredelmétodo*. El controlador es el método en el código de su función ejecutado por el motor de ejecución cuando se invoca la función.

En algunos idiomas, Lambda proporciona una biblioteca con una interfaz que espera que un método de controlador tenga un nombre específico. Para obtener información detallada sobre la nomenclatura de controladores para cada idioma, consulte los temas siguientes.

- [Creación de funciones de Lambda con Node.js](#)
- [Creación de funciones de Lambda con Python](#)
- [Creación de funciones de Lambda con Ruby](#)
- [Creación de funciones de Lambda con Java](#)
- [Creación de funciones de Lambda con Go](#)
- [Creación de funciones Lambda con C#](#)
- [Creación de funciones de Lambda con PowerShell](#)

## Lambda: error de conversión de la capa

Error: error de conversión de la capa de Lambda. Para obtener consejos sobre cómo resolver este problema, consulte la página [Solucionar problemas de implementación en Lambda](#) en la Guía del usuario de Lambda.

Al configurar una función de Lambda con una capa, Lambda la fusiona con el código de la función. Si este proceso no se completa, Lambda devuelve este error. Si encuentra este error, siga estos pasos:

- Eliminar cualquier archivo no utilizado de la capa
- Eliminar cualquier enlace simbólico de la capa
- Cambie el nombre de cualquier archivo que tenga el mismo nombre que un directorio en cualquiera de las capas de la función

## Lambda: InvalidParameterValueException o RequestEntityTooLargeException

Error: InvalidParameterValueException: Lambda no pudo configurar las variables de entorno porque las variables de entorno que ha proporcionado han excedido el límite de 4 KB. Cadena medida: {"A1":"uSFeY5cyPiPn7AtnX5BsM...

Error: RequestEntityTooLargeException: la solicitud debe ser menor de 5120 bytes para la operación UpdateFunctionConfiguration

El tamaño máximo del objeto de variables que se almacena en la configuración de la función no debe exceder los 4096 bytes. Esto incluye nombres clave, valores, comillas, comas y corchetes. El tamaño total del cuerpo de la solicitud HTTP también es limitado.

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs20.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Environment": {
    "Variables": {
      "BUCKET": "amzn-s3-demo-bucket",
      "KEY": "file.txt"
    }
  },
}
```

```
    ...  
}
```

En este ejemplo, el objeto tiene 39 caracteres y ocupa 39 bytes cuando se almacena (sin espacios en blanco) como la cadena {"BUCKET": "amzn-s3-demo-bucket", "KEY": "file.txt"}. Los caracteres ASCII estándar en valores de variables de entorno utilizan un byte cada uno. Los caracteres ASCII y Unicode extendidos pueden usar entre 2 y 4 bytes por carácter.

## Lambda: InvalidParameterValueException

Error: InvalidParameterValueException: Lambda no pudo configurar las variables de entorno porque las variables de entorno que ha proporcionado contienen claves reservadas que actualmente no se admiten para su modificación.

Lambda reserva algunas claves de variables de entorno para uso interno. Por ejemplo, el motor de ejecución usa `AWS_REGION` para determinar la región actual y no se puede reemplazar. Otras variables, como `PATH`, son usadas por el tiempo de ejecución, pero se pueden extender en la configuración de su función. Para obtener una lista completa, consulte [Variables definidas de entorno de tiempo de ejecución](#).

## Lambda: cuotas de simultaneidad y memoria

Error: ConcurrentExecutions especificada para la función reduce UnreservedConcurrentExecution de la cuenta por debajo de su valor mínimo

Error: el valor 'MemorySize' no cumplió con la restricción de seguridad: el miembro debe tener un valor menor o igual a 3008

Estos errores se producen cuando se supera la simultaneidad o las [cuotas](#) de memoria para su cuenta. Las nuevas cuentas de AWS tienen cuotas de simultaneidad y memoria reducidas. Para resolver errores relacionados a la simultaneidad, puede [solicitar un aumento de cuota](#). No puede solicitar aumentos de cuota.

- Simultaneidad: puede que se produzca un error si intenta crear una función mediante simultaneidad reservada o aprovisionada, o si la solicitud de simultaneidad por función ([PutFunctionConcurrency](#)) supera la cuota de simultaneidad de su cuenta.
- Memoria: se producen errores si la cantidad de memoria asignada a la función supera la cuota de memoria de la cuenta.

# Solucionar problemas de invocación en Lambda

Cuando invoca una función de Lambda, Lambda valida la solicitud y comprueba la capacidad de escalado antes de enviar el evento a su función o, para la invocación asíncrona, a la cola de eventos. Los errores de invocación pueden deberse a problemas con los parámetros de solicitud, la estructura de eventos, la configuración de funciones, los permisos de usuario, los permisos de recursos o los límites.

Si invoca su función directamente, verá errores de invocación en la respuesta de Lambda. Si invoca la función de forma asíncrona, con una asignación de origen de eventos o a través de otro servicio, es posible que encuentre errores en los registros, una cola de mensajes fallidos o un destino de evento fallido. Las opciones de manejo de errores y el comportamiento de reintento varían en función de cómo invoque la función y del tipo de error.

Para obtener una lista de los tipos de error que la operación `Invoke` puede regresar, consulte [Invocar](#).

## IAM: `lambda:InvokeFunction` no autorizado

```
Error: User: arn:aws:iam::123456789012:user/developer is not authorized to perform:
lambda:InvokeFunction on resource: my-function
```

El usuario o el rol que usted asume necesita permiso para invocar una función. Este requisito también se aplica a las funciones de Lambda y a otros recursos informáticos que invocan funciones. Agregue la política administrada de AWS `AWSLambdaRole` a su usuario o agregue una política personalizada que le permita la acción `lambda:InvokeFunction` en la función de destino.

### Note

El nombre de la acción de IAM (`lambda:InvokeFunction`) hace referencia a la operación de la API de Lambda `Invoke`.

Para obtener más información, consulte [Administrar permisos en AWS Lambda](#).

## Lambda: no se encontró un arranque válido (`Runtime.InvalidEntrypoint`)

```
Error: no se encontraron los arranques válidos: [/var/task/bootstrap/opt/bootstrap]
```

Este error suele ocurrir cuando la raíz del paquete de implementación no contiene un archivo ejecutable denominado `bootstrap`. Por ejemplo, si va a implementar una función de `provided.al2023` con un archivo `.zip`, el archivo de `bootstrap` debe estar en la raíz del archivo `.zip`, no en un directorio.

## Lambda: no se puede realizar la operación `ResourceConflictException`

Error: `ResourceConflictException`: la operación no se puede realizar en este momento. La función se encuentra actualmente en el siguiente estado: `Pending` (Pendiente)

Cuando conecta una función a una nube virtual privada (VPC) en el momento de la creación, la función entra en un estado `Pending` mientras Lambda crea interfaces de redes elásticas. Durante este tiempo, no puede invocar o modificar su función. Si conecta su función a una VPC después de la creación, puede invocarla mientras la actualización esté pendiente, pero no podrá modificar su código o configuración.

Para obtener más información, consulte [Estados de función de Lambda](#).

## Lambda: la función está atascada en Pendiente

Error: una función está bloqueada en el estado `Pending` durante varios minutos.

Si una función queda atascada en el estado `Pending` durante más de seis minutos, llame a una de las siguientes operaciones de API para desbloquearla.

- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)
- [PublishVersion](#)

Lambda cancela la operación pendiente y pone la función en el estado `Failed`. Luego, puede intentar realizar otra actualización.

## Lambda: una función utiliza toda la concurrencia

Problema: Una función utiliza toda la concurrencia disponible, lo que hace que otras funciones se limiten.

Para dividir la concurrencia disponible en una cuenta de AWS de una región de AWS en grupos, utilice [concurrencia reservada](#). La concurrencia reservada garantiza que una función siempre puede escalar a su concurrencia asignada, y que no escalará más allá de su concurrencia asignada.

## General: no se puede invocar la función con otras cuentas o servicios

Problema: puede invocar la función directamente, pero esta no se ejecuta cuando otro servicio o cuenta la invoca.

Puede conceder permisos a [otros servicios](#) y cuentas para invocar una función en la [política basada en recursos](#) de la función. Si el usuario que invoca la función está en otra cuenta, ese usuario también necesita [permiso para invocar funciones](#).

## General: la invocación de funciones está en bucle

Problema: la función se invoca continuamente en bucle.

Esto suele ocurrir cuando la función administra recursos en el mismo servicio de AWS que lo activa. Por ejemplo, es posible crear una función que almacene un objeto en un bucket de Amazon Simple Storage Service (Amazon S3) configurado con una [notificación que invoca la función de nuevo](#). Para evitar que la función se ejecute, reduzca la [simultaneidad](#) disponible a cero, lo que limita todas las invocaciones futuras. A continuación, identifique la ruta de acceso del código o el error de configuración que provocó la invocación recursiva. Lambda detecta y detiene de forma automática los bucles recursivos en algunos SDK y servicios de AWS. Para obtener más información, consulte [the section called “Detección de bucles recursivos”](#).

## Lambda: enrutamiento de alias con concurrencia aprovisionada

Problema: Invocaciones de casos de superación de simultaneidad aprovisionadas durante el enrutamiento de alias.

Lambda utiliza un modelo probabilístico simple para distribuir el tráfico entre las dos versiones de funciones. En niveles de tráfico bajos, es posible que vea una gran variación entre el porcentaje configurado y el porcentaje real de tráfico en cada versión. Si su función utiliza la concurrencia aprovisionada, puede evitar [Invocaciones de casos de superación](#) configurando un mayor número de instancias de simultaneidad aprovisionadas durante el tiempo en que el enrutamiento de alias está activo.


## Lambda: comienza en frío con concurrencia aprovisionada

Problema: Verá inicios en frío después de habilitar la concurrencia aprovisionada.



Cuando el número de ejecuciones simultáneas en una función es menor o igual que el [nivel configurado de concurrencia aprovisionada](#), no debería haber ningún comienzo frío. Para ayudarle a confirmar si la concurrencia aprovisionada funciona normalmente, haga lo siguiente:

- [Comprobar que la concurrencia aprovisionada esté habilitada](#) en la versión o alias de la función.

 Note

La simultaneidad aprovisionada no es compatible con la [versión no publicada de la función](#) (\$LATEST).

- Asegúrese de que los desencadenadores invoquen la versión o alias de función correctos. Por ejemplo, si está utilizando Amazon API Gateway, compruebe que API Gateway invoca la versión de la función o alias con concurrencia aprovisionada, no \$LATEST. Para confirmar que se está utilizando la concurrencia aprovisionada, puede comprobar la [métrica ProvisionedConcurrencyInvocations de Amazon CloudWatch](#). Un valor distinto de cero indica que la función está procesando invocaciones en entornos de ejecución inicializados.
- Determine si la concurrencia de funciones excede el nivel configurado de concurrencia aprovisionada comprobando la [métrica ProvisionedConcurrencySpilloverInvocations de CloudWatch](#). Un valor distinto de cero indica que toda la concurrencia aprovisionada está en uso y que se ha producido alguna invocación con un inicio en frío.
- Compruebe su [frecuencia de invocación](#) (solicitudes por segundo). Las funciones con concurrencia aprovisionada tienen una tasa máxima de 10 solicitudes por segundo por concurrencia aprovisionada. Por ejemplo, una función configurada con 100 concurrencia aprovisionada puede manejar 1000 solicitudes por segundo. Si la tasa de invocación supera las 1000 solicitudes por segundo, pueden producirse algunos inicios en frío.

## Lambda: el frío comienza con nuevas versiones

Problema: Verá arranques en frío al implementar nuevas versiones de su función.

Cuando actualiza un alias de función, Lambda cambia automáticamente la concurrencia aprovisionada a la nueva versión en función de los pesos configurados en el alias.

Error: `KMSDisabledException`: Lambda no pudo descifrar las variables de entorno porque la clave KMS utilizada está deshabilitada. Compruebe la configuración de claves de KMS de la función.

Este error puede producirse si su clave AWS Key Management Service (AWS KMS) está deshabilitada o si se revoca la concesión que permite a Lambda utilizar la clave. Si falta la concesión, configure la función para utilizar una clave diferente. Luego, reasigne la clave personalizada para volver a crear la concesión.

## EFS: la función no pudo montar el sistema de archivos EFS

Error: EFSMountFailureException: la función no pudo montar el sistema de archivos de EFS con el punto de acceso `arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd`.

Se rechazó la solicitud de montaje en el [sistema de archivos](#) de la función. Compruebe los permisos de la función y confirme que su sistema de archivos y su punto de acceso existen y están listos para su uso.

## EFS: la función no se pudo conectar al sistema de archivos EFS

Error: EFSMountConnectivityException: la función no se pudo conectar al sistema de archivos de Amazon EFS con el punto de acceso `arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd`. Compruebe la configuración de la red y vuelva a intentarlo.

La función no pudo establecer una conexión con el [sistema de archivos](#) de la función con el protocolo NFS (puerto TCP 2049). Compruebe el [grupo de seguridad y la configuración de enrutamiento](#) de las subredes de la VPC.

Si aparecen estos errores después de actualizar los ajustes de configuración de la VPC de la función, intente desmontar y volver a montar el sistema de archivos.

## EFS: La función no pudo montar el sistema de archivos EFS debido al tiempo de espera

Error: EFSMountTimeoutException: la función no pudo montar el sistema de archivos EFS con el punto de acceso `{arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd}` debido a que se agotó el tiempo de espera del montaje.

La función pudo conectarse al [sistema de archivos](#) de la función, pero se agotó el tiempo de espera de la operación de montaje. Vuelva a intentarlo después en un tiempo corto y considere limitar la [conurrencia](#) de la función para reducir la carga en el sistema de archivos.

## Lambda: Lambda detectó un proceso de E/S que estaba tomando demasiado tiempo

EFSIOException: esta instancia de función se detuvo porque Lambda detectó un proceso de E/S que estaba tardando demasiado.

Se agotó el tiempo de espera de una invocación anterior y Lambda no pudo finalizar el controlador de funciones. Este problema puede producirse cuando un sistema de archivos adjunto se queda sin créditos de ráfaga y el rendimiento previsto es insuficiente. Para aumentar el rendimiento, puede aumentar el tamaño del sistema de archivos o utilizar el rendimiento aprovisionado.

## Solucionar problemas de ejecución en Lambda

Cuando el tiempo de ejecución de Lambda ejecuta el código de la función, es posible que el evento se procese en una instancia de la función que ha estado procesando eventos durante algún tiempo, o que requiera que se inicialice una nueva instancia. Pueden producirse errores durante la inicialización de la función, cuando el código de controlador procesa el evento o cuando la función devuelve (o no devuelve) una respuesta.

Los errores de ejecución de funciones pueden deberse a problemas con el código, la configuración de funciones, los recursos descendentes o los permisos. Si invoca su función directamente, verá errores de función en la respuesta de Lambda. Si invoca la función de forma asíncrona, con una asignación de orígenes de eventos o a través de otro servicio, es posible que encuentre errores en los registros, una cola de mensajes fallidos o un destino en caso de error. Las opciones de manejo de errores y el comportamiento de reintento varían en función de cómo invoque la función y del tipo de error.

Cuando el código de función o el tiempo de ejecución de Lambda devuelven un error, el código de estado en la respuesta de Lambda es 200 OK. La presencia de un error en la respuesta se indica mediante un encabezado llamado `X-Amz-Function-Error`. Los códigos de estado de las series 400 y 500 están reservados para [errores de invocación](#).

## Lambda: la ejecución lleva demasiado tiempo

Problema: la ejecución de la función tarda demasiado tiempo.

Si el código tarda mucho más en ejecutarse en Lambda que en el equipo local, puede estar limitado por la memoria o la potencia de procesamiento disponibles para la función. [Configure la función con memoria adicional](#) para aumentar la memoria y la CPU.

## Lambda: los registros o rastros no aparecen

Problema: los registros no aparecen en los Registros de CloudWatch.

Problema: los registros de seguimiento no aparecen en AWS X-Ray.

La función necesita permiso para llamar a los Registros de CloudWatch y a X-Ray. Actualice su [rol de ejecución](#) para concederle permiso. Añada las siguientes políticas administradas para habilitar los registros y el seguimiento.

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

Cuando agregue permisos a su función, actualice también su código o configuración. Esto obliga a las instancias en ejecución de su función, cuyas credenciales han expirado, a detenerse y ser sustituidas.

### Note

Los registros pueden tardar de 5 a 10 minutos en aparecer después de una invocación de la función.

## Lambda: no aparecen todos los registros de mi función

Problema: faltan registros de funciones en Registros de CloudWatch, aunque mis permisos son correctos

Si su Cuenta de AWS alcanza los [límites de cuota de Registros de CloudWatch](#), CloudWatch limita el registro de funciones. Cuando esto sucede, es posible que algunos de los registros generados por sus funciones no aparezcan en los Registros de CloudWatch.

Si su función genera registros a una velocidad demasiado alta para que Lambda los procese, los resultados de los registros podrían no aparecer en Registros de CloudWatch. Cuando Lambda no puede enviar registros a CloudWatch a la velocidad a la que su función los produce, Lambda elimina los registros para evitar que la ejecución de la función se ralentice. Podrá observar de manera constante los registros descartados cuando el rendimiento del registro supere los 2 MB/s para un solo flujo de registro.

Si la función está configurada para usar [registros con formato JSON](#), Lambda intenta enviar un evento [logsDropped](#) a los registros de CloudWatch cuando los descarta. Sin embargo, cuando CloudWatch limita el registro de la función, es posible que este evento no llegue a los registros de CloudWatch, por lo que no siempre verá un registro cuando Lambda los descarte.

Para comprobar si su Cuenta de AWS alcanzó su límite de cuota de los Registros de CloudWatch, haga lo siguiente:

1. Abra la [consola de Service Quotas](#).
2. En el panel de navegación, elija Servicios de AWS.
3. En la lista de servicios de AWS, busque y seleccione Registros de Amazon CloudWatch.
4. En la lista de Service Quotas, seleccione las cuotas `CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions per second` y `PutLogEvents throttle limit in transactions per second` para ver su utilización.

También puede configurar alarmas de CloudWatch para que le avisen cuando el uso de su cuenta supere el límite que especifique para estas cuotas. Consulte [Creación de una alarma de CloudWatch basada en un umbral estático](#) para aprender más.

Si los límites de cuota predeterminados de Registros de CloudWatch no son suficientes para su caso, puede [solicitar un aumento de cuota](#).

## Lambda: la función regresa antes de que finalice la ejecución

Problema: (Node.js) La función se devuelve antes de que el código termine de ejecutarse

Muchas bibliotecas, incluido el AWS SDK, funcionan de forma asíncrona. Cuando realiza una llamada de red o lleva a cabo otra operación que requiere esperar una respuesta, las bibliotecas devuelven un objeto denominado promesa que realiza un seguimiento del progreso de la operación en segundo plano.

Para esperar a que la promesa se resuelva en una respuesta, utilice la palabra clave `await`. Esto bloquea la ejecución de su código de controlador hasta que la promesa se resuelve en un objeto que contiene la respuesta. Si no necesita usar los datos de la respuesta en el código, puede devolver la promesa directamente al tiempo de ejecución.

Algunas bibliotecas no devuelven promesas, pero se pueden envolver en código que sí lo hace. Para obtener más información, consulte [Definir el controlador de las funciones de Lambda en Node.js](#).

## AWS SDK: versiones y actualizaciones

Problema: El SDK de AWS incluido en el tiempo de ejecución no es la versión más reciente

Problema: el AWS SDK incluido en el tiempo de ejecución se actualiza en forma automática

Los tiempos de ejecución de los lenguajes de scripting incluyen el AWS SDK y se actualizan de manera periódica a la versión más reciente. La versión actual de cada tiempo de ejecución se muestra en la [página de tiempos de ejecución](#). Para utilizar una versión más reciente del AWS SDK o para bloquear las funciones a una versión específica, puede agrupar la biblioteca con el código de función o [crear una capa de Lambda](#). Para obtener información detallada sobre la creación de un paquete de implementación con dependencias, consulte los temas siguientes:

Node.js

[Implementar funciones Node.js de Lambda con archivos de archivo.zip](#)

Python

[Uso de archivos .zip para funciones de Lambda en Python](#)

Ruby

[Implementar funciones de Lambda de Ruby con archivos .zip](#)

Java

[Implementar funciones de Lambda Java con archivos de archivo .zip o JAR](#)

Go

[Implementar funciones de Lambda en Go con archivos .zip](#)

C#

[Crear e implementar funciones de Lambda C# con archivos de archivo .zip](#)

PowerShell

[Implemente funciones PowerShell Lambda con archivos de archivos.zip](#)

## Python: las bibliotecas se cargan de manera incorrecta

Problema: (Python) algunas bibliotecas no se cargan de manera correcta desde el paquete de implementación

Las bibliotecas con módulos de extensión escritos en C o C++ deben compilarse en un entorno con la misma arquitectura de procesador que Lambda (Amazon Linux). Para obtener más información, consulte [Uso de archivos .zip para funciones de Lambda en Python](#).

## Solucionar problemas de redes en Lambda

De forma predeterminada, Lambda ejecuta las funciones en una nube virtual privada (VPC) interna con conectividad a los servicios de AWS e Internet. Para acceder a los recursos de red local, puede [configurar la función para que se conecte a una VPC de la cuenta](#). Cuando utiliza esta característica, administra el acceso a Internet y la conectividad de red de la función con recursos de Amazon Virtual Private Cloud (Amazon VPC).

Los errores de conectividad de red pueden deberse a problemas de configuración de enrutamiento de la VPC, reglas de grupo de seguridad, permisos de rol de AWS Identity and Access Management (IAM), traducción de direcciones de red (NAT) o disponibilidad de recursos como direcciones IP o interfaces de red. En función del problema, es posible que aparezca un error específico o de tiempo de espera si una solicitud no puede alcanzar su destino.

### VPC: La función pierde acceso a Internet o agota el tiempo de espera

Problema: la función de Lambda pierde el acceso a Internet después de conectarse a una VPC.

Error: Error: Connect ETIMEDOUT 176.32.98.189:443

Error: Error: Tiempo de espera de la tarea agotado tras 10,00 segundos

Error: ReadTimeoutError: se agotó el tiempo de espera de lectura. (Tiempo de espera de lectura = 15)

Cuando conecta una función a una VPC, todas las solicitudes salientes pasan por la VPC. Para conectarse a Internet, configure la VPC para que envíe tráfico saliente desde la subred de la función a una puerta de enlace NAT en una subred pública. Para obtener más información y ejemplos de configuraciones de VPC, consulte [the section called "Acceso a Internet para funciones de VPC"](#).

Si se agota el tiempo de espera de algunas de las conexiones TCP, puede deberse a la fragmentación de paquetes. Las funciones de Lambda no pueden gestionar solicitudes TCP fragmentadas entrantes, ya que Lambda no admite la fragmentación de IP para TCP o ICMP.

### VPC: la función necesita acceso a los servicios de AWS sin utilizar internet

Problema: la función de Lambda necesita acceso a los servicios de AWS sin utilizar Internet.

Para conectar una función a servicios de AWS desde una subred privada sin acceso a Internet, utilice los puntos de conexión de VPC.

## VPC: se ha alcanzado el límite de la interfaz de red elástica

Error: ENILimitReachedException: Se alcanzó el límite de interfaz de red elástica para la VPC de la función.

Al conectar una función de Lambda a una VPC, Lambda crea una interfaz de red elástica para cada combinación de subred y grupo de seguridad adjuntos a la función. La cuota de servicio predeterminada es de 250 interfaces de red por VPC. Para solicitar un aumento de una cuota, use la [consola de Service Quotas](#).

## EC2: interfaz de red elástica con tipo de “lambda”

Código de error: Client.OperationNotPermitted

Mensaje de error: el grupo de seguridad no se puede modificar para este tipo de interfaz

Recibirá este error si intenta modificar una interfaz de red elástica (ENI) administrada por Lambda. No se incluye `ModifyNetworkInterfaceAttribute` en la API de Lambda para las operaciones de actualización en las interfaces de red elásticas creadas por Lambda.

## DNS: no se puede conectar a los hosts con UNKNOWNHOSTEXCEPTION

Mensaje de error: UNKNOWNHOSTEXCEPTION

Las funciones de Lambda admiten un máximo de 20 conexiones TCP simultáneas para la resolución de DNS. Es posible que la función esté agotando ese límite. Las solicitudes de DNS más comunes se llevan a cabo a través del UDP. Si la función solo establece conexiones DNS a través de UDP, es poco probable que este sea un problema. Este error suele producirse debido a una mala configuración o a una infraestructura degradada, por lo que antes de examinar en profundidad el tráfico de DNS, confirme que su infraestructura de DNS está correctamente configurada y en buen estado y que la función de Lambda haga referencia a un host especificado en el DNS.

Si diagnostica que el problema está relacionado con el límite máximo de conexión TCP, tenga en cuenta que no puede solicitar un aumento de este límite. Si su función de Lambda recurre al DNS de TCP debido a las grandes cargas útiles de DNS, confirme que la solución utilice bibliotecas compatibles con EDNS. Para obtener más información acerca de EDNS, consulte el [estándar](#)



---

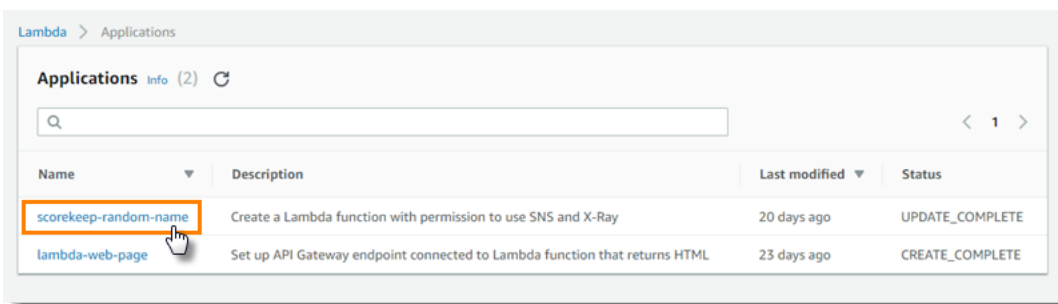
[RFC 6891](#). Si las cargas de DNS superan constantemente los tamaños máximos de EDNS, es posible que la solución siga agotando el límite de DNS de TCP.

# Visualización de aplicaciones en la consola de AWS Lambda

La consola de AWS Lambda le ayuda a monitorizar y administrar sus aplicaciones de Lambda. La sección Aplicaciones muestra las pilas de AWS CloudFormation con funciones de Lambda. El menú incluye pilas que lanza en AWS CloudFormation mediante la consola de AWS CloudFormation, el AWS Serverless Application Repository, la AWS CLI o la CLI de AWS SAM.

Para ver una aplicación de Lambda

1. Abra la página [Applications \(Aplicaciones\)](#) de la consola de Lambda.
2. Elija una aplicación.



La información general muestra la siguiente información sobre la aplicación.

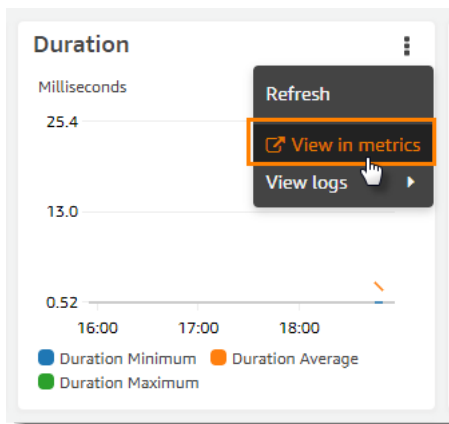
- Plantilla AWS CloudFormation o Plantilla SAM: la plantilla que define su aplicación.
- Recursos: los recursos de AWS que se definen en la plantilla de su aplicación. Para administrar las funciones de Lambda de su aplicación, elija un nombre de función de la lista.

# Monitoreo de aplicaciones de Lambda

La sección Aplicaciones de la consola de Lambda incluye la pestaña Monitoreo en la que puede revisar un panel de Amazon CloudWatch con métricas agrupadas de los recursos de su aplicación.

Para monitorizar una aplicación de Lambda

1. Abra la página [Applications \(Aplicaciones\)](#) de la consola de Lambda.
2. Elija Monitoring (Monitorización).
3. Para ver más detalles sobre las métricas de cualquier gráfico, seleccione Ver en métricas en el menú desplegable.



El gráfico aparece en una nueva pestaña, con las métricas pertinentes que se indican debajo del gráfico. Puede personalizar la vista de este gráfico, cambiando las métricas y los recursos mostrados, la estadística, el periodo y otros factores para obtener una mejor comprensión de la situación actual.

De forma predeterminada, la consola de Lambda muestra un panel básico. Puede personalizar esta página agregando uno o más paneles de Amazon CloudWatch a su plantilla de aplicación con el tipo de recurso [AWS::CloudWatch::Dashboard](#). Si su plantilla incluye uno o varios paneles, la página le mostrará sus paneles en lugar del panel predeterminado. Puede cambiar entre los paneles con el menú desplegable en la parte superior derecha de la página. En el siguiente ejemplo se crea un panel con un único widget que crea gráficos del número de invocaciones de una función denominada `my-function`.

Example plantilla de panel de funciones

Resources :

```
MyDashboard:
  Type: AWS::CloudWatch::Dashboard
  Properties:
    DashboardName: my-dashboard
    DashboardBody: |
      {
        "widgets": [
          {
            "type": "metric",
            "width": 12,
            "height": 6,
            "properties": {
              "metrics": [
                [
                  "AWS/Lambda",
                  "Invocations",
                  "FunctionName",
                  "my-function",
                  {
                    "stat": "Sum",
                    "label": "MyFunction"
                  }
                ],
                [
                  {
                    "expression": "SUM(METRICS())",
                    "label": "Total Invocations"
                  }
                ]
              ],
              "region": "us-east-1",
              "title": "Invocations",
              "view": "timeSeries",
              "stacked": false
            }
          }
        ]
      }
```

Para obtener más información acerca de la creación de paneles y widgets de CloudWatch, consulte [Estructura y sintaxis del cuerpo del panel](#) en la Referencia de la API de Amazon CloudWatch.

# Creación de implementaciones continuas para las funciones de Lambda

Utilice implementaciones continuas para controlar los riesgos asociados a la introducción de nuevas versiones de su función de Lambda. En una implementación continua, el sistema implementa automáticamente la nueva versión de la función y envía gradualmente una cantidad creciente de tráfico a la nueva versión. La cantidad de tráfico y la tasa de aumento son parámetros que puede configurar.

Puede configurar una implementación sucesiva mediante el uso de AWS CodeDeploy y AWS SAM. CodeDeploy es un servicio que automatiza las implementaciones de aplicaciones en plataformas informáticas de Amazon como Amazon EC2 y AWS Lambda. Para obtener más información, consulte [¿Qué es CodeDeploy?](#). Al utilizar CodeDeploy para implementar la función de Lambda, puede monitorizar fácilmente el estado de la implementación e iniciar una restauración si detecta algún problema.

AWS SAM es un marco de código abierto para crear aplicaciones sin servidor. Cree una plantilla de AWS SAM (en formato YAML) para especificar la configuración de los componentes necesarios para la implementación continua. AWS SAM utiliza la plantilla para crear y configurar los componentes. Para obtener más información, consulte [¿Qué es AWS SAM?](#)

En una implementación continua, AWS SAM realiza las siguientes tareas:

- Configura su función de Lambda y crea un alias.

La configuración de direccionamiento de alias es la capacidad subyacente que implementa la implementación continua.

- Crea una aplicación CodeDeploy y un grupo de implementación.

El grupo de implementación administra la implementación continua y la restauración (si es necesario).

- Detecta cuando crea una nueva versión de su función de Lambda.
- Activa CodeDeploy para iniciar la implementación de la nueva versión.

## Ejemplo AWS SAM de la plantilla Lambda

En el siguiente ejemplo se muestra una [plantilla de AWS SAM](#) para una implementación continua simple.

```
AWSTemplateFormatVersion : '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Description: A sample SAM template for deploying Lambda functions.  
  
Resources:  
# Details about the myDateTimeFunction Lambda function  
  myDateTimeFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: myDateTimeFunction.handler  
      Runtime: nodejs18.x  
# Creates an alias named "live" for the function, and automatically publishes when you  
  update the function.  
    AutoPublishAlias: live  
    DeploymentPreference:  
# Specifies the deployment configuration  
    Type: Linear10PercentEvery2Minutes
```

Esta plantilla define una función de Lambda denominada `myDateTimeFunction` con las siguientes propiedades.

### AutoPublishAlias

La propiedad `AutoPublishAlias` crea un alias denominado `live`. Además, el marco de AWS SAM detecta automáticamente cuando guarda código nuevo para la función. A continuación, el marco publica una nueva versión de función y actualiza el alias `live` para que apunte a la nueva versión.

### DeploymentPreference

La propiedad `DeploymentPreference` determina la velocidad a la que la aplicación CodeDeploy cambia el tráfico de la versión original de la función de Lambda a la nueva versión. El valor `Linear10PercentEvery2Minutes` desplaza un diez por ciento adicional del tráfico a la nueva versión cada dos minutos.

Para obtener una lista de las configuraciones de implementación predefinidas, consulte [Configuraciones de implementación](#).

Para obtener un tutorial detallado sobre cómo utilizar CodeDeploy con funciones de Lambda, consulte [Implementación de una función de Lambda actualizada con CodeDeploy](#).

# Uso de Lambda con Kubernetes

Puede implementar y administrar las funciones de Lambda con la API de Kubernetes mediante los [Controladores para Kubernetes \(ACK\) de AWS](#) o [Crossplane](#).

## Controladores para Kubernetes (ACK) de AWS

Puede utilizar ACK para implementar y administrar recursos de AWS de la API de Kubernetes. A través de ACK, AWS proporciona controladores personalizados de código abierto para AWS servicios como Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) y Amazon SageMaker. Cada servicio de AWS compatible tiene su propio controlador personalizado. En su clúster de Kubernetes, instale un controlador para cada servicio de AWS que desee utilizar. Luego, cree una [Definición de recursos personalizada \(CRD\)](#) para definir los recursos de AWS.

Se recomienda que utilice [Helm 3.8 o posterior](#) a fin de instalar los controladores para ACK. Cada controlador para ACK viene con su propio gráfico de Helm, que instala el controlador, los CRD y las reglas RBAC de Kubernetes. Para obtener más información, consulte [Instalar un controlador para ACK](#) en la documentación de ACK.

Después de crear el recurso personalizado de ACK, puede utilizarlo como cualquier otro objeto de Kubernetes integrado. Por ejemplo, puede implementar y administrar las funciones de Lambda con las cadenas de herramientas de Kubernetes que prefiera, como [kubectl](#).

Estos son algunos ejemplos de casos de uso para el aprovisionamiento de funciones de Lambda a través de ACK:

- Su organización utiliza el [control de acceso basado en roles \(RBAC\)](#) y [roles de IAM para cuentas de servicio](#) a fin de crear límites de permisos. Con ACK, puede reutilizar este modelo de seguridad para Lambda sin tener que crear políticas ni usuarios nuevos.
- Su organización tiene un DevOps proceso para implementar recursos en un clúster de Amazon Elastic Kubernetes Service (Amazon EKS) mediante manifiestos de Kubernetes. Con ACK, puede utilizar un manifiesto para aprovisionar funciones de Lambda sin tener que crear una infraestructura independiente en forma de plantillas de código.

Para obtener más información sobre el uso de ACK, consulte el [Tutorial de Lambda en la documentación de ACK](#).



# Crossplane

[Crossplane](#) es un proyecto de código abierto de la Cloud Native Computing Foundation (CNCF) que utiliza Kubernetes para administrar los recursos de la infraestructura en la nube. Con Crossplane, los desarrolladores pueden solicitar infraestructura sin necesidad de entender sus complejidades. Los equipos de plataformas mantienen el control sobre cómo se aprovisiona y administra la infraestructura.

Con Crossplane, puede implementar y administrar las funciones de Lambda con sus cadenas de herramientas de Kubernetes preferidas, como [kubecti](#) y cualquier canalización de CI/CD que pueda implementar manifiestos en Kubernetes. Estos son algunos ejemplos de casos de uso para el aprovisionamiento de funciones de Lambda a través de Crossplane:

- Su organización quiere garantizar el cumplimiento de las normas al garantizar que las funciones de Lambda tengan las [etiquetas](#) correctas. Los equipos de plataformas pueden utilizar [Composiciones de Crossplane](#) para definir esta política mediante abstracciones de API. Luego, los desarrolladores pueden utilizar estas abstracciones para implementar funciones de Lambda con etiquetas.
- Su proyecto lo usa con Kubernetes. GitOps En este modelo, Kubernetes reconcilia continuamente el repositorio de git (estado deseado) con los recursos que se ejecutan dentro del clúster (estado actual). Si hay diferencias, el GitOps proceso realiza cambios automáticamente en el clúster. [Puede usarlo GitOps con Kubernetes para implementar y administrar funciones de Lambda a través de Crossplane, utilizando herramientas y conceptos familiares de Kubernetes, como los CRD y los controladores.](#)

Para obtener más información sobre el uso de Crossplane con Lambda, consulte lo siguiente:

- [Esquemas para Crossplane de AWS](#): este repositorio incluye ejemplos de cómo utilizar Crossplane para implementar recursos de AWS, incluidas las funciones de Lambda.

## Note

Los Esquemas para Crossplane de AWS se encuentran en fase de desarrollo activo y no deberían utilizarse en la producción.

- [Implementación de Lambda con Amazon EKS y Crossplane](#): en este video se muestra un ejemplo avanzado de implementación de una arquitectura sin servidor de AWS con Crossplane, que explora el diseño desde la perspectiva del desarrollador y de la plataforma.

# Aplicaciones de muestra de Lambda

El repositorio de GitHub para esta guía contiene aplicaciones de ejemplo en las que se muestra el uso de diferentes lenguajes y servicios de AWS. Cada una de las aplicaciones de ejemplo contiene scripts para facilitar la implementación, la limpieza y los recursos de soporte.

## Node.js

### Aplicaciones de Lambda de ejemplo en Node.js

- [blank-nodejs](#): una función de Node.js que muestra el uso de registro, las variables de entorno, el seguimiento de AWS X-Ray, las capas, las pruebas de unidad y el AWS SDK.
- [nodejs-apig](#): una función con un punto de conexión de API público que procesa un evento desde API Gateway y devuelve una respuesta HTTP.
- [efs-nodejs](#): una función que utiliza un sistema de archivos de Amazon EFS en una Amazon VPC. Este ejemplo incluye una VPC, un sistema de archivos, objetivos de montaje y un punto de acceso configurados para su uso con Lambda.

## Python

### Aplicaciones de ejemplo de Lambda de ejemplo en Python

- [blank-ruby](#): una función de Python que muestra el uso del registro, las variables de entorno, el seguimiento de AWS X-Ray, las pruebas de unidad y el AWS SDK.

## Ruby

### Aplicaciones de Lambda de ejemplo en Ruby

- [blank-ruby](#): una función de Ruby que muestra el uso del registro, las variables de entorno, el seguimiento de AWS X-Ray, las pruebas de unidad y el AWS SDK.
- [Ejemplos de código de Ruby para AWS Lambda](#): muestras de código escritas en Ruby que demuestran cómo interactuar con AWS Lambda.

## Java

### Aplicaciones de Lambda de ejemplo en Java

- [java17-examples](#): una función de Java que demuestra cómo utilizar un registro de Java para representar un objeto de datos de eventos de entrada.
- [java-basic](#): una colección de funciones de Java mínimas con pruebas unitarias y configuración de registro variable.
- [java-events](#): una colección de funciones Java que contiene un código básico sobre cómo gestionar los eventos de varios servicios, como Amazon API Gateway, Amazon SQS y Amazon Kinesis. Estas funciones utilizan la última versión de la biblioteca [aws-lambda-java-events](#) (3.0.0 y más recientes). Estos ejemplos no requieren utilizar AWS SDK como una dependencia.
- [s3-java](#): una función de Java que procesa los eventos de notificación de Amazon S3 y utiliza Java Class Library (JCL) para crear miniaturas de los archivos de imagen cargados.
- [Uso de API Gateway para invocar una función de Lambda](#): una función Java que escanea una tabla de Amazon DynamoDB que contiene información sobre los empleados. Luego, utiliza Amazon Simple Notification Service para enviar un mensaje de texto a los empleados que celebran sus aniversarios laborales. En este ejemplo, se utiliza API Gateway para invocar la función.

### Ejecución de marcos Java populares en Lambda

- [spring-cloud-function-samples](#): un ejemplo de Spring que muestra cómo utilizar el marco [Spring Cloud Function](#) para crear funciones de Lambda AWS.
- [Demostración de la aplicación Spring Boot sin servidor](#): un ejemplo que muestra cómo configurar una aplicación Spring Boot típica en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de GraalVM con un tiempo de ejecución personalizado.
- [Demostración de la aplicación Micronaut sin servidor](#): un ejemplo que muestra cómo usar Micronaut en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de GraalVM con un tiempo de ejecución personalizado. Obtenga más información en las [guías de Micronaut/Lambda](#).
- [Demostración de la aplicación Quarkus sin servidor](#): un ejemplo que muestra cómo usar Quarkus en un tiempo de ejecución Java gestionado con y sin SnapStart, o como una imagen nativa de GraalVM con un tiempo de ejecución personalizado. [Obtenga más información en la guía de Quarkus/Lambda y en la guía de Quarkus/SnapStart.](#)

## Go

Lambda proporciona las siguientes aplicaciones de ejemplo para el tiempo de ejecución Go:

### Aplicaciones de ejemplo de Lambda en Go

- [go-al2](#): una función de “Hola, mundo” que devuelve la dirección IP pública. Esta aplicación utiliza el tiempo de ejecución personalizado `provided.al2`.
- [blank-go](#): una función Go que muestra el uso de las bibliotecas de Go de Lambda, el registro, las variables de entorno y el AWS SDK. Esta aplicación utiliza el tiempo de ejecución `go1.x`.

## C#

### Aplicaciones de muestra de Lambda en C#

- [blank-csharp](#): una función de C# que muestra el uso de las bibliotecas de .NET de Lambda, el registro, las variables de entorno, el seguimiento de AWS X-Ray, las pruebas de unidad y el SDK de AWS.
- [blank-csharp-with-layer](#): una función de C# que utiliza la CLI de .NET para crear una capa que empaquete las dependencias de la función.
- [ec2-spot](#): una función que administra las solicitudes de instancia de spot en Amazon EC2.

## PowerShell

Lambda proporciona las siguientes aplicaciones de muestra para PowerShell:

- [blank-powershell](#): una función de PowerShell que muestra el uso de registro, las variables de entorno y el AWS SDK.

Para implementar una aplicación de ejemplo, siga las instrucciones de su archivo README.

# Utilización de Lambda con SDK de AWS

Los kits de desarrollo de software (SDK) de AWS se encuentran disponibles en muchos lenguajes de programación populares. Cada SDK proporciona una API, ejemplos de código y documentación que facilitan a los desarrolladores la creación de aplicaciones en su lenguaje preferido.

Documentación de SDK	Ejemplos de código
<a href="#">AWS SDK for C++</a>	<a href="#">Ejemplos de código de AWS SDK for C++</a>
<a href="#">AWS CLI</a>	<a href="#">Ejemplos de código de AWS CLI</a>
<a href="#">AWS SDK for Go</a>	<a href="#">Ejemplos de código de AWS SDK for Go</a>
<a href="#">AWS SDK for Java</a>	<a href="#">Ejemplos de código de AWS SDK for Java</a>
<a href="#">AWS SDK for JavaScript</a>	<a href="#">Ejemplos de código de AWS SDK for JavaScript</a>
<a href="#">AWS SDK para Kotlin</a>	<a href="#">Ejemplos de código de AWS SDK para Kotlin</a>
<a href="#">AWS SDK for .NET</a>	<a href="#">Ejemplos de código de AWS SDK for .NET</a>
<a href="#">AWS SDK for PHP</a>	<a href="#">Ejemplos de código de AWS SDK for PHP</a>
<a href="#">AWS Tools for PowerShell</a>	<a href="#">Ejemplos de código de Herramientas para PowerShell</a>
<a href="#">AWS SDK for Python (Boto3)</a>	<a href="#">Ejemplos de código de AWS SDK for Python (Boto3)</a>
<a href="#">AWS SDK for Ruby</a>	<a href="#">Ejemplos de código de AWS SDK for Ruby</a>
<a href="#">AWS SDK para Rust</a>	<a href="#">Ejemplos de código de AWS SDK para Rust</a>
<a href="#">AWS SDK para SAP ABAP</a>	<a href="#">Ejemplos de código de AWS SDK para SAP ABAP</a>
<a href="#">AWS SDK para Swift</a>	<a href="#">Ejemplos de código de AWS SDK para Swift</a>

Para ver ejemplos específicos de Lambda, consulte [Ejemplos de código de Lambda con SDK de AWS](#).

 Ejemplo de disponibilidad

¿No encuentra lo que necesita? Solicite un ejemplo de código a través del enlace de Enviar comentarios que se encuentra al final de esta página.

# Ejemplos de código de Lambda con SDK de AWS

Los siguientes ejemplos de código muestran cómo utilizar Lambda con un kit de desarrollo de software (SDK) de AWS.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las funciones de servicio individuales, es posible ver las acciones en contexto en los escenarios relacionados y en los ejemplos entre servicios.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica llamando a varias funciones dentro del mismo servicio.

Los ejemplos con varios servicios son aplicaciones de muestra que funcionan con varios Servicios de AWS.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Introducción

## Introducción a Lambda

En los siguientes ejemplos de código se muestra cómo empezar a utilizar Lambda.

.NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
namespace LambdaActions;  
  
using Amazon.Lambda;  
  
public class HelloLambda
```

```
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {

            Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for .NET.

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Código del archivo de CMake CMakeLists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)

# Set this project's name.
```



```
project("hello_lambda")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
  libraries for the AWS SDK.
  string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
    "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
  list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
  # Copy relevant AWS SDK for C++ libraries into the current binary directory
  for running and debugging.

  # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
  may need to uncomment this
  # and set the proper subdirectory to the
  executables' location.

  AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
    ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
  hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
  ${AWSSDK_LINK_LIBRARIES})
```

Código del archivo de origen hello\_lambda.cpp.

```
#include <aws/core/Aws.h>
#include <aws/lambda/LambdaClient.h>
```

```
#include <aws/lambda/model/ListFunctionsRequest.h>
#include <iostream>

/*
 * A "Hello Lambda" starter application which initializes an AWS Lambda (Lambda)
 client and lists the Lambda functions.
 *
 * main function
 *
 * Usage: 'hello_lambda'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
    // options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Lambda::LambdaClient lambdaClient(clientConfig);
        std::vector<Aws::String> functions;
        Aws::String marker; // Used for pagination.

        do {
            Aws::Lambda::Model::ListFunctionsRequest request;
            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::Lambda::Model::ListFunctionsOutcome outcome =
lambdaClient.ListFunctions(
                request);

            if (outcome.IsSuccess()) {
                const Aws::Lambda::Model::ListFunctionsResult
&listFunctionsResult = outcome.GetResult();
                std::cout << listFunctionsResult.GetFunctions().size()
                    << " lambda functions were retrieved." << std::endl;
            }
        }
    }
}
```

```

        for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: listFunctionsResult.GetFunctions()) {
            functions.push_back(functionConfiguration.GetFunctionName());
            std::cout << functions.size() << " "
                << functionConfiguration.GetDescription() <<
std::endl;

            std::cout << " "
                <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                functionConfiguration.GetRuntime()) << ": "
                << functionConfiguration.GetHandler()
                << std::endl;
        }
        marker = listFunctionsResult.GetNextMarker();
    } else {
        std::cerr << "Error with Lambda::ListFunctions. "
            << outcome.GetError().GetMessage()
            << std::endl;
        result = 1;
        break;
    }
} while (!marker.empty());
}

Aws::ShutdownAPI(options); // Should only be called once.
return result;
}

```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for C++.

Go

SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up
// to 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    lambdaClient := lambda.NewFromConfig(sdkConfig)

    maxItems := 10
    fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
    result, err := lambdaClient.ListFunctions(context.TODO(),
&lambda.ListFunctionsInput{
    MaxItems: aws.Int32(int32(maxItems)),
})
    if err != nil {
        fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
        return
    }
    if len(result.Functions) == 0 {
        fmt.Println("You don't have any functions!")
    } else {
        for _, function := range result.Functions {
            fmt.Printf("\t\t%v\n", *function.FunctionName)
        }
    }
}
```

```
}  
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for Go.

## Java

### SDK para Java 2.x

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
package com.example.lambda;  
  
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.lambda.LambdaClient;  
import software.amazon.awssdk.services.lambda.model.LambdaException;  
import software.amazon.awssdk.services.lambda.model.ListFunctionsResponse;  
import software.amazon.awssdk.services.lambda.model.FunctionConfiguration;  
import java.util.List;  
  
/**  
 * Before running this Java V2 code example, set up your development  
 * environment, including your credentials.  
 *  
 * For more information, see the following documentation topic:  
 *  
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html  
 */  
public class ListLambdaFunctions {  
    public static void main(String[] args) {  
        Region region = Region.US_WEST_2;  
        LambdaClient awsLambda = LambdaClient.builder()  
            .region(region)
```

```
        .build();

        listFunctions(awsLambda);
        awsLambda.close();
    }

    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
                System.out.println("The function name is " +
config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for Java 2.x.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
```

```
const paginator = paginateListFunctions({ client }, {});
const functions = [];

for await (const page of paginator) {
  const funcNames = page.Functions.map((f) => f.FunctionName);
  functions.push(...funcNames);
}

console.log("Functions:");
console.log(functions.join("\n"));
return functions;
};
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for JavaScript.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import boto3

def main():
    """
    List the Lambda functions in your AWS account.
    """
    # Create the Lambda client
    lambda_client = boto3.client("lambda")

    # Use the paginator to list the functions
    paginator = lambda_client.get_paginator("list_functions")
    response_iterator = paginator.paginate()
```

```
print("Here are the Lambda functions in your account:")
for page in response_iterator:
    for function in page["Functions"]:
        print(f" {function['FunctionName']}")

if __name__ == "__main__":
    main()
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
require 'aws-sdk-lambda'

# Creates an AWS Lambda client using the default credentials and configuration
def lambda_client
  Aws::Lambda::Client.new
end

# Lists the Lambda functions in your AWS account, paginating the results if
# necessary
def list_lambda_functions
  lambda = lambda_client

  # Use a pagination iterator to list all functions
  functions = []
  lambda.list_functions.each_page do |page|
```



```
functions.concat(page.functions)
end

# Print the name and ARN of each function
functions.each do |function|
  puts "Function name: #{function.function_name}"
  puts "Function ARN: #{function.function_arn}"
  puts
end

puts "Total functions: #{functions.count}"
end

list_lambda_functions if __FILE__ == $PROGRAM_NAME
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for Ruby.

## Ejemplos de código

- [Acciones de Lambda utilizando SDK de AWS](#)
  - [Uso de CreateAlias con un AWS SDK o la CLI](#)
  - [Uso de CreateFunction con un AWS SDK o la CLI](#)
  - [Uso de DeleteAlias con un AWS SDK o la CLI](#)
  - [Uso de DeleteFunction con un AWS SDK o la CLI](#)
  - [Uso de DeleteFunctionConcurrency con un AWS SDK o la CLI](#)
  - [Uso de DeleteProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
  - [Uso de GetAccountSettings con un AWS SDK o la CLI](#)
  - [Uso de GetAlias con un AWS SDK o la CLI](#)
  - [Uso de GetFunction con un AWS SDK o la CLI](#)
  - [Uso de GetFunctionConcurrency con un AWS SDK o la CLI](#)
  - [Uso de GetFunctionConfiguration con un AWS SDK o la CLI](#)
  - [Uso de GetPolicy con un AWS SDK o la CLI](#)
  - [Uso de GetProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
  - [Uso de Invoke con un AWS SDK o la CLI](#)

- [Uso de ListFunctions con un AWS SDK o la CLI](#)
- [Uso de ListProvisionedConcurrencyConfigs con un AWS SDK o la CLI](#)
- [Uso de ListTags con un AWS SDK o la CLI](#)
- [Uso de ListVersionsByFunction con un AWS SDK o la CLI](#)
- [Uso de PublishVersion con un AWS SDK o la CLI](#)
- [Uso de PutFunctionConcurrency con un AWS SDK o la CLI](#)
- [Uso de PutProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
- [Uso de RemovePermission con un AWS SDK o la CLI](#)
- [Uso de TagResource con un AWS SDK o la CLI](#)
- [Uso de UntagResource con un AWS SDK o la CLI](#)
- [Uso de UpdateAlias con un AWS SDK o la CLI](#)
- [Uso de UpdateFunctionCode con un AWS SDK o la CLI](#)
- [Uso de UpdateFunctionConfiguration con un AWS SDK o la CLI](#)
- [Escenarios de Lambda con SDK de AWS](#)
  - [Confirmación de manera automática a los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK](#)
  - [Migración en forma automática los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK](#)
  - [Primeros pasos para la creación y la invocación de funciones de Lambda con un AWS SDK](#)
  - [Escriba datos de actividad personalizados con una función de Lambda tras la autenticación de usuarios de Amazon Cognito mediante un AWS SDK](#)
- [Ejemplos sin servidor para Lambda que utilizan SDK de AWS](#)
  - [Conexión a una base de datos de Amazon RDS en una función de Lambda](#)
  - [Invocar una función de Lambda desde un desencadenador de Kinesis](#)
  - [Invocación de una función de Lambda desde un desencadenador de DynamoDB](#)
  - [Invocación de una función de Lambda desde un desencadenador de Amazon DocumentDB](#)
  - [Invocación de una función de Lambda desde un desencadenador de Amazon MSK](#)
  - [Invocación de una función de Lambda desde un desencadenador de Amazon S3](#)
  - [Invocar una función de Lambda desde un desencadenador de Amazon SNS](#)
  - [Invocar una función de Lambda desde un desencadenador de Amazon SQS](#)

- [Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis](#)
- [Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB](#)
- [Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.](#)
- [Ejemplos de servicios combinados de Lambda con SDK de AWS](#)
  - [Creación de una API de REST de API Gateway para realizar un seguimiento de datos de COVID-19](#)
  - [Creación de una API de REST de biblioteca de préstamos](#)
  - [Creación de una aplicación de mensajería con Step Functions](#)
  - [Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas](#)
  - [Creación una aplicación de chat de websocket con API Gateway](#)
  - [Creación de una aplicación que analice los comentarios de los clientes y sintetice el audio](#)
  - [Invocación de una función Lambda desde un navegador](#)
  - [Transformación de datos para su aplicación con S3 Object Lambda](#)
  - [Uso de API Gateway para invocar una función de Lambda](#)
  - [Uso de Step Functions para invocar funciones de Lambda](#)
  - [Uso de eventos programados para invocar una función de Lambda](#)

## Acciones de Lambda utilizando SDK de AWS

Los siguientes ejemplos de código muestran cómo llevar a cabo acciones individuales de Lambda con los SDK de AWS. Estos fragmentos llaman a la API de Lambda y son fragmentos de código de programas de gran tamaño que deben ejecutarse en contexto. En cada ejemplo se incluye un enlace a GitHub, con instrucciones de configuración y ejecución del código.

Los siguientes ejemplos incluyen solo las acciones que se utilizan con mayor frecuencia. Para ver una lista completa, consulte la [Referencia de la API de AWS Lambda](#).

### Ejemplos

- [Uso de CreateAlias con un AWS SDK o la CLI](#)
- [Uso de CreateFunction con un AWS SDK o la CLI](#)

- [Uso de DeleteAlias con un AWS SDK o la CLI](#)
- [Uso de DeleteFunction con un AWS SDK o la CLI](#)
- [Uso de DeleteFunctionConcurrency con un AWS SDK o la CLI](#)
- [Uso de DeleteProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
- [Uso de GetAccountSettings con un AWS SDK o la CLI](#)
- [Uso de GetAlias con un AWS SDK o la CLI](#)
- [Uso de GetFunction con un AWS SDK o la CLI](#)
- [Uso de GetFunctionConcurrency con un AWS SDK o la CLI](#)
- [Uso de GetFunctionConfiguration con un AWS SDK o la CLI](#)
- [Uso de GetPolicy con un AWS SDK o la CLI](#)
- [Uso de GetProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
- [Uso de Invoke con un AWS SDK o la CLI](#)
- [Uso de ListFunctions con un AWS SDK o la CLI](#)
- [Uso de ListProvisionedConcurrencyConfigs con un AWS SDK o la CLI](#)
- [Uso de ListTags con un AWS SDK o la CLI](#)
- [Uso de ListVersionsByFunction con un AWS SDK o la CLI](#)
- [Uso de PublishVersion con un AWS SDK o la CLI](#)
- [Uso de PutFunctionConcurrency con un AWS SDK o la CLI](#)
- [Uso de PutProvisionedConcurrencyConfig con un AWS SDK o la CLI](#)
- [Uso de RemovePermission con un AWS SDK o la CLI](#)
- [Uso de TagResource con un AWS SDK o la CLI](#)
- [Uso de UntagResource con un AWS SDK o la CLI](#)
- [Uso de UpdateAlias con un AWS SDK o la CLI](#)
- [Uso de UpdateFunctionCode con un AWS SDK o la CLI](#)
- [Uso de UpdateFunctionConfiguration con un AWS SDK o la CLI](#)

## Uso de **CreateAlias** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar CreateAlias.

## CLI

### AWS CLI

#### Creación de un alias para una función de Lambda

En el siguiente ejemplo de `create-alias`, se crea un alias llamado LIVE que apunta a la versión 1 de la función de Lambda `my-function`.

```
aws lambda create-alias \  
  --function-name my-function \  
  --description "alias for live version of function" \  
  --function-version 1 \  
  --name LIVE
```

Salida:

```
{  
  "FunctionVersion": "1",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",  
  "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",  
  "Description": "alias for live version of function"  
}
```

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [CreateAlias](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo crea un nuevo alias de Lambda para una versión y configuración de enrutamiento específicas a fin de especificar el porcentaje de solicitudes de invocación que recibe.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"} -Description "Alias
for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Para obtener información sobre la API, consulte [CreateAlias](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **CreateFunction** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `CreateFunction`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

### .NET

#### AWS SDK for .NET

##### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
```

```
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //           the source code is stored.
    // S3Key     - The name of the file containing the code.
    var functionCode = new FunctionCode
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };


    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for .NET.

## C++

## SDK para C++

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::CreateFunctionRequest request;
request.SetFunctionName(LAMBDA_NAME);
request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
    request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
    request.SetTimeout(15);
    request.SetMemorySize(128);

    // Assume the AWS Lambda function was built in Docker with same
    architecture
    // as this code.
    #if defined(__x86_64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
    #elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
    #else
        #error "Unimplemented architecture"
    #endif // defined(architecture)
    #else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_8);
    #endif

    request.SetRole(roleArn);
    request.SetHandler(LAMBDA_HANDLER_NAME);
    request.SetPublish(true);
    Aws::Lambda::Model::FunctionCode code;
```



```
std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                      std::ios_base::in | std::ios_base::binary);
if (!ifstream.is_open()) {
    std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;

#ifdef USE_CPP_LAMBDA_FUNCTION
    std::cerr
        << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
        << std::endl;
#endif

    deleteIamRole(clientConfig);
    return false;
}

Aws::StringStream buffer;
buffer << ifstream.rdbuf();

code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                      buffer.str().length()));

request.SetCode(code);

Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
    request);

if (outcome.IsSuccess()) {
    std::cout << "The lambda function was successfully created. " <<
seconds
        << " seconds elapsed." << std::endl;
    break;
}

else {
    std::cerr << "Error with CreateFunction. "
        << outcome.GetError().GetMessage()
        << std::endl;
    deleteIamRole(clientConfig);
    return false;
}
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

Para crear una función de Lambda

El siguiente ejemplo de `create-function` crea una función de Lambda con el nombre `my-function`.

```
aws lambda create-function \  
  --function-name my-function \  
  --runtime nodejs18.x \  
  --zip-file fileb://my-function.zip \  
  --handler my-function.handler \  
  --role arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-tges6bf4
```

Contenidos de `my-function.zip`:

```
This file is a deployment package that contains your function code and any dependencies.
```

Salida:

```
{  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "PFn4S+er27qk+UuZSTKEQfNKG/XNn7QJs90mJgq6oH8=",  
  "FunctionName": "my-function",  
  "CodeSize": 308,  
  "RevisionId": "873282ed-4cd3-4dc8-a069-d0c647e470c6",  
  "MemorySize": 128,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
  "Version": "$LATEST",  
  "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-zgur6bf4",  
  "Timeout": 3,  
}
```

```
"LastModified": "2023-10-14T22:26:11.234+0000",
"Handler": "my-function.handler",
"Runtime": "nodejs18.x",
"Description": ""
}
```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información de la API, consulte [CreateFunction](#) en la Referencia de comandos de la AWS CLI.

## Go

### SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// CreateFunction creates a new Lambda function from code contained in the
// zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
// until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(functionName string, handlerName
    string,
```

```


iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
var state types.State
_, err := wrapper.LambdaClient.CreateFunction(context.TODO(),
&lambda.CreateFunctionInput{
    Code:          &types.FunctionCode{ZipFile: zipPackage.Bytes()},
    FunctionName:  aws.String(functionName),
    Role:          iamRoleArn,
    Handler:       aws.String(handlerName),
    Publish:       true,
    Runtime:       types.RuntimePython38,
})
if err != nil {
    var resConflict *types.ResourceConflictException
    if errors.As(err, &resConflict) {
        log.Printf("Function %v already exists.\n", functionName)
        state = types.StateActive
    } else {
        log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
    }
} else {
    waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
    funcOutput, err := waiter.WaitForOutput(context.TODO(),
&lambda.GetFunctionInput{
        FunctionName: aws.String(functionName)}, 1*time.Minute)
    if err != nil {
        log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
}
return state
}

```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for Go.

## Java

## SDK para Java 2.x

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.CreateFunctionRequest;
import software.amazon.awssdk.services.lambda.model.FunctionCode;
import software.amazon.awssdk.services.lambda.model.CreateFunctionResponse;
import software.amazon.awssdk.services.lambda.model.GetFunctionRequest;
import software.amazon.awssdk.services.lambda.model.GetFunctionResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;
import software.amazon.awssdk.services.lambda.model.Runtime;
import software.amazon.awssdk.services.lambda.waiters.LambdaWaiter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

/**
 * This code example requires a ZIP or JAR that represents the code of the
 * Lambda function.
 * If you do not have a ZIP or JAR, please refer to the following document:
 *
 * https://github.com/aws-doc-sdk-examples/tree/master/javav2/usecases/
creating_workflows_stepfunctions
 *
 * Also, set up your development environment, including your credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class CreateFunction {
    public static void main(String[] args) {

        final String usage = ""

            Usage:
                <functionName> <filePath> <role> <handler>\s

            Where:
                functionName - The name of the Lambda function.\s
                filePath - The path to the ZIP or JAR where the code is
located.\s
                role - The role ARN that has Lambda permissions.\s
                handler - The fully qualified method name (for example,
example.Handler::handleRequest). \s
            """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        String filePath = args[1];
        String role = args[2];
        String handler = args[3];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
            .region(region)
            .build();

        createLambdaFunction(awsLambda, functionName, filePath, role, handler);
        awsLambda.close();
    }

    public static void createLambdaFunction(LambdaClient awsLambda,
        String functionName,
        String filePath,
        String role,
        String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            InputStream is = new FileInputStream(filePath);
```

```
    SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

    FunctionCode code = FunctionCode.builder()
        .zipFile(fileToUpload)
        .build();

    CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
        .functionName(functionName)
        .description("Created by the Lambda Java API")
        .code(code)
        .handler(handler)
        .runtime(Runtime.JAVA8)
        .role(role)
        .build();

    // Create a Lambda function using a waiter.
    CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
    GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
        .functionName(functionName)
        .build();
    WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
    waiterResponse.matched().response().ifPresent(System.out::println);
    System.out.println("The function ARN is " +
functionResponse.functionArn());

    } catch (LambdaException | FileNotFoundException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
}
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for Java 2.x.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for JavaScript.

## Kotlin

### SDK para Kotlin

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).



```
suspend fun createNewFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String? {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }

    val request =
        CreateFunctionRequest {
            functionName = myFunctionName
            code = functionCode
            description = "Created by the Lambda Kotlin API"
            handler = myHandler
            role = myRole
            runtime = Runtime.Java8
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitForFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn
    }
}
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la Referencia de la API de AWS SDK para Kotlin.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function createFunction($functionName, $role, $bucketName, $handler)
{
    //This assumes the Lambda function is in an S3 bucket.
    return $this->customWaiter(function () use ($functionName, $role,
$bucketName, $handler) {
        return $this->lambdaClient->createFunction([
            'Code' => [
                'S3Bucket' => $bucketName,
                'S3Key' => $functionName,
            ],
            'FunctionName' => $functionName,
            'Role' => $role['Arn'],
            'Runtime' => 'python3.9',
            'Handler' => "$handler.lambda_handler",
        ]);
    });
}
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for PHP.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se crea una nueva función de C# (tiempo de ejecución de dotnetcore1.0) denominada myFunction en AWS Lambda, que proporciona los binarios compilados para la función desde un archivo zip del sistema de archivos local (se pueden utilizar rutas relativas o absolutas). Las funciones de Lambda C# especifican el controlador de la función mediante la designación `AssemblyName::Namespace.ClassName::MethodName`.

Debe reemplazar adecuadamente las partes del nombre del ensamblado (sin el sufijo .dll), el espacio de nombres, el nombre de la clase y el nombre del método de la especificación del controlador. La nueva función tendrá las variables de entorno 'envvar1' y 'envvar2' configuradas a partir de los valores proporcionados.

```
Publish-LMFunction -Description "My C# Lambda Function" `
  -FunctionName MyFunction `
  -ZipFilename .\MyFunctionBinaries.zip `
  -Handler "AssemblyName::Namespace.ClassName::MethodName" `
  -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
  -Runtime dotnetcore1.0 `
  -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

Salida:

```
CodeSha256      : /NgBmd...gq71I=
CodeSize       : 214784
DeadLetterConfig :
Description    : My C# Lambda Function
Environment    : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn    : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName   : MyFunction
Handler        : AssemblyName::Namespace.ClassName::MethodName
KMSKeyArn     :
LastModified   : 2016-12-29T23:50:14.207+0000
MemorySize    : 128
Role           : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime        : dotnetcore1.0
Timeout        : 3
Version        : $LATEST
VpcConfig     :
```

Ejemplo 2: este ejemplo es similar al anterior, excepto que los binarios de la función se cargan primero en un bucket de Amazon S3 (que debe estar en la misma región que la función de Lambda prevista) y, a continuación, se hace referencia al objeto de S3 resultante al crear la función.

```
Write-S3Object -BucketName mybucket -Key MyFunctionBinaries.zip -File .
  \MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
  -FunctionName MyFunction `
```

```
-BucketName mybucket `
-Key MyFunctionBinaries.zip `
-Handler "AssemblyName::Namespace.ClassName::MethodName" `
-Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
-Runtime dotnetcore1.0 `
-Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- Para obtener información acerca de la API, consulte [CreateFunction](#) en la Referencia de cmdlets de AWS Tools for PowerShell.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
        This
                               must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
        function
                               code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
```

```
"""
try:
    response = self.lambda_client.create_function(
        FunctionName=function_name,
        Description="AWS Lambda doc example",
        Runtime="python3.8",
        Role=iam_role.arn,
        Handler=handler_name,
        Code={"ZipFile": deployment_package},
        Publish=True,
    )
    function_arn = response["FunctionArn"]
    waiter = self.lambda_client.get_waiter("function_active_v2")
    waiter.wait(FunctionName=function_name)
    logger.info(
        "Created function '%s' with ARN: '%s'.",
        function_name,
        response["FunctionArn"],
    )
except ClientError:
    logger.error("Couldn't create function %s.", function_name)
    raise
else:
    return function_arn
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
```

```
attr_accessor :lambda_client

def initialize
  @lambda_client = Aws::Lambda::Client.new
  @logger = Logger.new($stdout)
  @logger.level = Logger::WARN
end

# Deploys a Lambda function.
#
# @param function_name: The name of the Lambda function.
# @param handler_name: The fully qualified name of the handler function. This
#                       must include the file name and the function name.
# @param role_arn: The IAM role to use for the function.
# @param deployment_package: The deployment package that contains the function
#                             code in .zip format.
# @return: The Amazon Resource Name (ARN) of the newly created function.
def create_function(function_name, handler_name, role_arn, deployment_package)
  response = @lambda_client.create_function({
    role: role_arn.to_s,
    function_name: function_name,
    handler: handler_name,
    runtime: "ruby2.7",
    code: {
      zip_file: deployment_package
    },
    environment: {
      variables: {
        "LOG_LEVEL" => "info"
      }
    }
  })

  @lambda_client.wait_until(:function_active_v2, { function_name:
function_name}) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  response
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error creating #{function_name}:\n #{e.message}")
rescue Aws::Writers::Errors::WaiterFailed => e
  @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
end
```

- Para obtener información sobre la API, consulte [CreateFunction](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/**
 * Create a function, uploading from a zip file.
 */
pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
    let code = self.prepare_function(zip_file, None).await?;

    let key = code.s3_key().unwrap().to_string();

    let role = self.create_role().await.map_err(|e| anyhow!(e))?;

    info!("Created iam role, waiting 15s for it to become active");
    tokio::time::sleep(Duration::from_secs(15)).await;

    info!("Creating lambda function {}", self.lambda_name);
    let _ = self
        .lambda_client
        .create_function()
        .function_name(self.lambda_name.clone())
        .code(code)
        .role(role.arn())
        .runtime(aws_sdk_lambda::types::Runtime::ProvidedAl2)
        .handler("_unused")
        .send()
        .await
        .map_err(anyhow::Error::from)?;
```

```
self.wait_for_function_ready().await?;

self.lambda_client
    .publish_version()
    .function_name(self.lambda_name.clone())
    .send()
    .await?;

Ok(key)
}

/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder()
        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build())
}
```



- Para obtener información sobre la API, consulte [CreateFunction](#) en la Referencia de la API de AWS SDK para Rust.

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
TRY.
  lo_lmd->createfunction(
    iv_functionname = iv_function_name
    iv_runtime = `python3.9`
    iv_role = iv_role_arn
    iv_handler = iv_handler
    io_code = io_zip_file
    iv_description = 'AWS Lambda code example'
  ).
  MESSAGE 'Lambda function created.' TYPE 'I'.
  CATCH /aws1/cx_lmdcodesigningcfn00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
  CATCH /aws1/cx_lmdcodestorageexcdex.
    MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
  CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
  CATCH /aws1/cx_lmdinvalidcodesigex.
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
  CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
  CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
  CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
  CATCH /aws1/cx_lmdserviceexception.
```

```
MESSAGE 'An internal problem was encountered by the AWS Lambda service.'  
TYPE 'E'.  
CATCH /aws1/cx_lmdtoomanyrequestsex.  
MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.  
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [CreateFunction](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **DeleteAlias** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar DeleteAlias.

### CLI

#### AWS CLI

Eliminación de un alias de función de Lambda

En el siguiente ejemplo de `delete-alias`, se elimina el alias nombrado LIVE de la función de Lambda `my-function`.

```
aws lambda delete-alias \  
  --function-name my-function \  
  --name LIVE
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [DeleteTopic](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se elimina el alias de la función de Lambda mencionado en el comando.

```
Remove-LMAlias -FunctionName "MyLambdaFunction123" -Name "NewAlias"
```

- Para obtener información sobre la API, consulte [DeleteAlias](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de `DeleteFunction` con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteFunction`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

### .NET

#### AWS SDK for .NET

##### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// Delete an AWS Lambda function.  
/// </summary>  
/// <param name="functionName">The name of the Lambda function to
```

```
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</
returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for .NET.

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::DeleteFunctionRequest request;
```

```
request.SetFunctionName(LAMBDA_NAME);

Aws::Lambda::Model::DeleteFunctionOutcome outcome = client.DeleteFunction(
    request);

if (outcome.IsSuccess()) {
    std::cout << "The lambda function was successfully deleted." <<
std::endl;
}
else {
    std::cerr << "Error with Lambda::DeleteFunction. "
    << outcome.GetError().GetMessage()
    << std::endl;
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

Ejemplo 1: eliminar una función de Lambda con el nombre de la función

En el siguiente ejemplo de `delete-function`, se elimina la función de Lambda denominada `my-function` especificando el nombre de la función.

```
aws lambda delete-function \  
  --function-name my-function
```

Este comando no genera ninguna salida.

Ejemplo 2: eliminar una función de Lambda con el ARN de la función

En el siguiente ejemplo de `delete-function`, se elimina la función de Lambda denominada `my-function` especificando el ARN de la función.

```
aws lambda delete-function \  
  --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Este comando no genera ninguna salida.

### Ejemplo 3: eliminar una función de Lambda con el ARN parcial de la función

En el siguiente ejemplo de `delete-function`, se elimina la función de Lambda denominada `my-function` especificando el ARN parcial de la función.

```
aws lambda delete-function \  
  --function-name 123456789012:function:my-function
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información de la API, consulte [DeleteFunction](#) en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// FunctionWrapper encapsulates function actions used in the examples.  
// It contains an AWS Lambda service client that is used to perform user actions.  
type FunctionWrapper struct {  
  LambdaClient *lambda.Client  
}  
  
// DeleteFunction deletes the Lambda function specified by functionName.  
func (wrapper FunctionWrapper) DeleteFunction(functionName string) {  
  _, err := wrapper.LambdaClient.DeleteFunction(context.TODO(),  
    &lambda.DeleteFunctionInput{  
      FunctionName: aws.String(functionName),  
    })  
}
```

```
if err != nil {
    log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
}
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for Go.

## Java

### SDK para Java 2.x

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.DeleteFunctionRequest;
import software.amazon.awssdk.services.lambda.model.LambdaException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class DeleteFunction {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <functionName>\s
```

```
        Where:
            functionName - The name of the Lambda function.\s
            """";

    if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String functionName = args[0];
    Region region = Region.US_EAST_1;
    LambdaClient awsLambda = LambdaClient.builder()
        .region(region)
        .build();

    deleteLambdaFunction(awsLambda, functionName);
    awsLambda.close();
}

public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
    try {
        DeleteFunctionRequest request = DeleteFunctionRequest.builder()
            .functionName(functionName)
            .build();

        awsLambda.deleteFunction(request);
        System.out.println("The " + functionName + " function was deleted");

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for Java 2.x.



## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for JavaScript.

## Kotlin

### SDK para Kotlin

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
suspend fun delLambdaFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
```

```
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la Referencia de la API de AWS SDK para Kotlin.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function deleteFunction($functionName)
{
    return $this->lambdaClient->deleteFunction([
        'FunctionName' => $functionName,
    ]);
}
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for PHP.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo elimina una versión específica de una función de Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Deletes a Lambda function.
  # @param function_name: The name of the function to delete.
  def delete_function(function_name)
    print "Deleting function: #{function_name}..."
    @lambda_client.delete_function(
      function_name: function_name
    )
    print "Done!".green
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
  end
end
```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
pub async fn delete_function(
    &self,
    location: Option<String>,
) -> (
    Result<DeleteFunctionOutput, anyhow::Error>,
    Result<DeleteRoleOutput, anyhow::Error>,
    Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
    info!("Deleting lambda function {}", self.lambda_name);
    let delete_function = self
        .lambda_client
        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    info!("Deleting iam role {}", self.role_name);
    let delete_role = self
        .iam_client
        .delete_role()
        .role_name(self.role_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
        if let Some(location) = location {
            info!("Deleting object {location}");
            Some(
```

```

        self.s3_client
            .delete_object()
            .bucket(self.bucket.clone())
            .key(location)
            .send()
            .await
            .map_err(anyhow::Error::from),
    )
} else {
    info!(?location, "Skipping delete object");
    None
};

(delete_function, delete_role, delete_object)
}

```

- Para obtener información sobre la API, consulte [DeleteFunction](#) en la Referencia de la API de AWS SDK para Rust.

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

TRY.
    lo_lmd->deletefunction( iv_functionname = iv_function_name ).
    MESSAGE 'Lambda function deleted.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.

```

```
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
    CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [DeleteFunction](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de `DeleteFunctionConcurrency` con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteFunctionConcurrency`.

### CLI

#### AWS CLI

Eliminación del límite de ejecución simultánea reservado de una función

En el siguiente ejemplo de `delete-function-concurrency`, se elimina el límite de ejecución simultánea reservado de la función `my-function`.

```
aws lambda delete-function-concurrency \
  --function-name my-function
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Reserva de simultaneidad para una función de Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [DeleteFunctionConcurrency](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo elimina la simultaneidad de funciones de la función de Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MyLambdaFunction123"
```

- Para obtener información sobre la API, consulte [DeleteFunctionConcurrency](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **DeleteProvisionedConcurrencyConfig** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteProvisionedConcurrencyConfig`.

### CLI

#### AWS CLI

Eliminación de la configuración de simultaneidad aprovisionada

En el siguiente ejemplo de `delete-provisioned-concurrency-config`, se elimina la configuración de simultaneidad aprovisionada para el alias GREEN de la función especificada.

```
aws lambda delete-provisioned-concurrency-config \  
  --function-name my-function \  
  --qualifier GREEN
```

- Para obtener información sobre la API, consulte [DeleteProvisionedConcurrencyConfig](#) en la Referencia de comandos de la AWS CLI.



## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se elimina la configuración de simultaneidad aprovisionada para un alias específico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MyLambdaFunction123" -
Qualifier "NewAlias1"
```

- Para obtener información sobre la API, consulte [DeleteProvisionedConcurrencyConfig](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetAccountSettings** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetAccountSettings`.

### CLI

#### AWS CLI

Recuperación de detalles de su cuenta en una región de AWS

En el siguiente ejemplo de `get-account-settings`, se muestran los límites de Lambda y la información de uso de su cuenta.

```
aws lambda get-account-settings
```

Salida:

```
{
  "AccountLimit": {
    "CodeSizeUnzipped": 262144000,
    "UnreservedConcurrentExecutions": 1000,
    "ConcurrentExecutions": 1000,
    "CodeSizeZipped": 52428800,
    "TotalCodeSize": 80530636800
  }
}
```

```

    },
    "AccountUsage": {
      "FunctionCount": 4,
      "TotalCodeSize": 9426
    }
  }
}

```

Para obtener más información, consulte [Límites de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [GetAccountSettings](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo, se muestra para comparar el límite de la cuenta y el uso de la cuenta

```

Get-LMAccountSetting | Select-Object
@{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
@{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}

```

Salida:

```

TotalCodeSizeLimit TotalCodeSizeUsed
-----
80530636800      15078795

```

- Para obtener información sobre la API, consulte [GetAccountSettings](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetAlias** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetAlias`.

## CLI

### AWS CLI

Recuperación de detalles sobre el alias de una función

En el siguiente ejemplo de `get-alias`, se muestran detalles del alias nombrado `LIVE` de la función de Lambda `my-function`.

```
aws lambda get-alias \  
  --function-name my-function \  
  --name LIVE
```

Salida:

```
{  
  "FunctionVersion": "3",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",  
  "RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",  
  "Description": "alias for live version of function"  
}
```

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [GetAlias](#) en la Referencia de comandos de AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo, se recuperan las ponderaciones de Routing Config para un alias de función de Lambda específico.

```
Get-LMAlias -FunctionName "MyLambdaFunction123" -Name "newlabel1" -Select  
RoutingConfig
```

Salida:

```
AdditionalVersionWeights
-----
{[1, 0.6]}
```

- Para obtener información sobre la API, consulte [GetAlias](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetFunction** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetFunction`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
{
```

```
var functionRequest = new GetFunctionRequest
{
    FunctionName = functionName,
};

var response = await _lambdaService.GetFunctionAsync(functionRequest);
return response.Configuration;
}
```

- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for .NET.

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::GetFunctionRequest request;
request.SetFunctionName(functionName);

Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

if (outcome.IsSuccess()) {
    std::cout << "Function retrieve.\n" <<
outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
```

```
        << std::endl;
    }
    else {
        std::cerr << "Error with Lambda::GetFunction. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
}
```

- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

Cómo recuperar información sobre una función

En el siguiente ejemplo de `get-function` se muestra información sobre la función `my-function`.

```
aws lambda get-function \
  --function-name my-function
```

Salida:

```
{
  "Concurrency": {
    "ReservedConcurrentExecutions": 100
  },
  "Code": {
    "RepositoryType": "S3",
    "Location": "https://awslambda-us-west-2-tasks.s3.us-west-2.amazonaws.com/snapshots/123456789012/my-function..."
  },
  "Configuration": {
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Version": "$LATEST",
    "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJm1KidWaaCgk=",

```


```
    "FunctionName": "my-function",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "MemorySize": 128,
    "RevisionId": "28f0fb31-5c5c-43d3-8955-03e76c5c1075",
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::123456789012:role/service-role/helloWorldPython-
role-uy3l9qyq",
    "Timeout": 3,
    "LastModified": "2019-09-24T18:20:35.054+0000",
    "Runtime": "nodejs10.x",
    "Description": ""
}
}
```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información sobre la API, consulte [GetFunction](#) en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
```

```
LambdaClient *lambda.Client
}

// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(functionName string) types.State {
    var state types.State
    funcOutput, err := wrapper.LambdaClient.GetFunction(context.TODO(),
        &lambda.GetFunctionInput{
            FunctionName: aws.String(functionName),
        })
    if err != nil {
        log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
    return state
}
```

- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for Go.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const getFunction = (funcName) => {
    const client = new LambdaClient({});
    const command = new GetFunctionCommand({ FunctionName: funcName });
    return client.send(command);
};
```



- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for JavaScript.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function getFunction($functionName)
{
    return $this->lambdaClient->getFunction([
        'FunctionName' => $functionName,
    ]);
}
```

- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for PHP.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource
```

```
def get_function(self, function_name):
    """
    Gets data about a Lambda function.

    :param function_name: The name of the function.
    :return: The function data.
    """
    response = None
    try:
        response =
self.lambda_client.get_function(FunctionName=function_name)
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            logger.info("Function %s does not exist.", function_name)
        else:
            logger.error(
                "Couldn't get function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    return response
```

- Para obtener información sobre la API, consulte [GetFunction](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
```

```

attr_accessor :lambda_client

def initialize
  @lambda_client = Aws::Lambda::Client.new
  @logger = Logger.new($stdout)
  @logger.level = Logger::WARN
end

# Gets data about a Lambda function.
#
# @param function_name: The name of the function.
# @return response: The function data, or nil if no such function exists.
def get_function(function_name)
  @lambda_client.get_function(
    {
      function_name: function_name
    }
  )
rescue Aws::Lambda::Errors::ResourceNotFoundException => e
  @logger.debug("Could not find function: #{function_name}:\n #{e.message}")
  nil
end

```

- Para obtener información sobre la API, consulte [GetFunction](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
    info!("Getting lambda function");

```

```

        self.lambda_client
            .get_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

```

- Para obtener información sobre la API, consulte [GetFunction](#) en la Referencia de la API de AWS SDK para Rust.

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

TRY.
    oo_result = lo_lmd->getfunction( iv_functionname = iv_function_name ).
    " oo_result is returned for testing purposes. "
    MESSAGE 'Lambda function information retrieved.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.

```

- Para obtener detalles sobre la API, consulte [GetFunction](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de `GetFunctionConcurrency` con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetFunctionConcurrency`.

### CLI

#### AWS CLI

Visualización de la configuración de simultaneidad reservada para una función

En el siguiente ejemplo de `get-function-concurrency`, se recupera la configuración de simultaneidad reservada para la función especificada.

```
aws lambda get-function-concurrency \  
  --function-name my-function
```

Salida:

```
{  
  "ReservedConcurrentExecutions": 250  
}
```

- Para obtener información sobre la API, consulte [GetFunctionConcurrency](#) en la Referencia de comandos de la AWS CLI.

### PowerShell

#### Herramientas para PowerShell

Ejemplo 1: este ejemplo obtiene la simultaneidad reservada para la función de Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Salida:

```
ReservedConcurrentExecutions
```

```
-----  
100
```

- Para obtener información sobre la API, consulte [GetFunctionConcurrency](#) en la Referencia de Cmdlet AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetFunctionConfiguration** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetFunctionConfiguration`.

### CLI

#### AWS CLI

Recuperación de la configuración específica de la versión de una función de Lambda

En el siguiente ejemplo de `get-function-configuration`, se muestran los ajustes de la versión 2 de la función `my-function`.

```
aws lambda get-function-configuration \  
  --function-name my-function:2
```

Salida:

```
{  
  "FunctionName": "my-function",  
  "LastModified": "2019-09-26T20:28:40.438+0000",  
  "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",  
  "MemorySize": 256,  
  "Version": "2",  
  "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-uy319qq",  
  "Timeout": 3,  
  "Runtime": "nodejs10.x",  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
}
```

```

"CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJm1KidWoaCgk=",
>Description": "",
>VpcConfig": {
>  "SubnetIds": [],
>  "VpcId": "",
>  "SecurityGroupIds": []
>,
> "CodeSize": 304,
> "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
> "Handler": "index.handler"
}

```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información sobre la API, consulte [GetFunctionConfiguration](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo devuelve la configuración específica de la versión de una función de Lambda.

```

Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
"PowershellAlias"

```

### Salida:

```

CodeSha256           : uW0W0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize             : 1426
DeadLetterConfig     : Amazon.Lambda.Model.DeadLetterConfig
Description          : Verson 3 to test Aliases
Environment          : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn          : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                    :PowershellAlias
FunctionName         : MylambdaFunction123
Handler              : lambda_function.launch_instance
KMSKeyArn            :

```

```
LastModified           : 2019-12-25T09:52:59.872+0000
LastUpdateStatus      : Successful
LastUpdateStatusReason :
LastUpdateStatusReasonCode :
Layers                 : {}
MasterArn              :
MemorySize             : 128
RevisionId             : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                   : arn:aws:iam::123456789012:role/service-role/lambda
Runtime                : python3.8
State                  : Active
StateReason            :
StateReasonCode        :
Timeout                : 600
TracingConfig          : Amazon.Lambda.Model.TracingConfigResponse
Version                : 4
VpcConfig              : Amazon.Lambda.Model.VpcConfigDetail
```

- Para obtener información sobre la API, consulte [GetFunctionConfiguration](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetPolicy** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetPolicy`.

### CLI

#### AWS CLI

Recuperación de la política de IAM basada en recursos para una función, una versión o un alias

En el siguiente ejemplo de `get-policy` se muestra información de la política sobre la función de Lambda `my-function`.

```
aws lambda get-policy \  
  --function-name my-function
```



**Salida:**

```
{
  "Policy": {
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [
      {
        "Sid": "iot-events",
        "Effect": "Allow",
        "Principal": {"Service": "iotevents.amazonaws.com"},
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-
function"
      }
    ],
  },
  "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668"
}
```

Para obtener más información, consulte [Uso de políticas basadas en recursos para AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [GetPolicy](#) en la Referencia de comandos de la AWS CLI.

**PowerShell****Herramientas para PowerShell**

Ejemplo 1: en este ejemplo se muestra la política de funciones de la función de Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

**Salida:**

```
{"Version": "2012-10-17", "Id": "default", "Statement":
[{"Sid": "xxxx", "Effect": "Allow", "Principal":
{"Service": "sns.amazonaws.com"}, "Action": "lambda:InvokeFunction", "Resource": "arn:aws:lamb
east-1:123456789102:function:test"]}]}
```

- Para obtener información sobre la API, consulte [GetPolicy](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **GetProvisionedConcurrencyConfig** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetProvisionedConcurrencyConfig`.

### CLI

#### AWS CLI

Visualización de una configuración de simultaneidad aprovisionada

En el siguiente ejemplo de `get-provisioned-concurrency-config` se muestran detalles de la configuración de simultaneidad aprovisionada para el alias BLUE de la función especificada.

```
aws lambda get-provisioned-concurrency-config \  
  --function-name my-function \  
  --qualifier BLUE
```

Salida:

```
{  
  "RequestedProvisionedConcurrentExecutions": 100,  
  "AvailableProvisionedConcurrentExecutions": 100,  
  "AllocatedProvisionedConcurrentExecutions": 100,  
  "Status": "READY",  
  "LastModified": "2019-12-31T20:28:49+0000"  
}
```

- Para obtener información sobre la API, consulte [GetProvisionedConcurrencyConfig](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo obtiene la configuración de simultaneidad aprovisionada para el alias especificado de la función de Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MyLambdaFunction123" -
Qualifier "NewAlias1"
```

#### Salida:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                    : IN_PROGRESS
StatusReason                              :
```

- Para obtener información sobre la API, consulte [GetProvisionedConcurrencyConfig](#) en la Referencia de Cmdlets de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **Invoke** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar Invoke.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

## .NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).


```
/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {
        FunctionName = functionName,
        Payload = payload,
    };

    var response = await _lambdaService.InvokeAsync(request);
    MemoryStream stream = response.Payload;
    string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
    return returnValue;
}
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for .NET.

## C++

## SDK para C++

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::InvokeRequest request;
request.SetFunctionName(LAMBDA_NAME);
request.SetLogType(logType);
std::shared_ptr<Aws::IOStream> payload =
Aws::MakeShared<Aws::StringStream>(
    "FunctionTest");
*payload << jsonPayload.View().WriteReadable();
request.SetBody(payload);
request.SetContentType("application/json");
Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

if (outcome.IsSuccess()) {
    invokeResult = std::move(outcome.GetResult());
    result = true;
    break;
}

else {
    std::cerr << "Error with Lambda::InvokeRequest. "
              << outcome.GetError().GetMessage()
              << std::endl;
    break;
}
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

Ejemplo 1: invocar una función de Lambda de forma sincrónica

En el siguiente ejemplo de `invoke`, se invoca la función `my-function` de forma sincrónica. La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la CLI de AWS. Para obtener más información, consulte [las opciones globales de la línea de comandos admitidas en la CLI de AWS](#) en la Guía del usuario de la interfaz de línea de comandos de AWS.

```
aws lambda invoke \  
  --function-name my-function \  
  --cli-binary-format raw-in-base64-out \  
  --payload '{ "name": "Bob" }' \  
  response.json
```

Salida:

```
{  
  "ExecutedVersion": "$LATEST",  
  "StatusCode": 200  
}
```

Para obtener más información, consulte [Invocación sincrónica](#) en la Guía para desarrolladores de Lambda de AWS.

Ejemplo 2: invocar una función de Lambda de forma asíncrona

En el siguiente ejemplo de `invoke`, se invoca la función `my-function` de forma asíncrona. La opción `cli-binary-format` es obligatoria si va a utilizar la versión 2 de la CLI de AWS. Para obtener más información, consulte [las opciones globales de la línea de comandos admitidas en la CLI de AWS](#) en la Guía del usuario de la interfaz de línea de comandos de AWS.

```
aws lambda invoke \  

```

```
--function-name my-function \  
--invocation-type Event \  
--cli-binary-format raw-in-base64-out \  
--payload '{ "name": "Bob" }' \  
response.json
```

Salida:

```
{  
  "statusCode": 202  
}
```

Para obtener más información, consulte [Invocación asíncrona](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información sobre la API, consulte [Invocar](#) en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// FunctionWrapper encapsulates function actions used in the examples.  
// It contains an AWS Lambda service client that is used to perform user actions.  
type FunctionWrapper struct {  
  LambdaClient *lambda.Client  
}  
  
// Invoke invokes the Lambda function specified by functionName, passing the  
// parameters  
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which  
// tells
```

```
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(functionName string, parameters any, getLog
bool) *lambda.InvokeOutput {
    logType := types.LogTypeNone
    if getLog {
        logType = types.LogTypeTail
    }
    payload, err := json.Marshal(parameters)
    if err != nil {
        log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
    }
    invokeOutput, err := wrapper.LambdaClient.Invoke(context.TODO(),
&lambda.InvokeInput{
        FunctionName: aws.String(functionName),
        LogType:      logType,
        Payload:      payload,
    })
    if err != nil {
        log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
    }
    return invokeOutput
}
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for Go.

## Java

### SDK para Java 2.x

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
import org.json.JSONObject;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lambda.LambdaClient;
```



```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
import software.amazon.awssdk.services.lambda.model.LambdaException;

public class LambdaInvoke {

    /**
     * Function names appear as
     * arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
     * you can retrieve the value by looking at the function in the AWS Console
     *
     * Also, set up your development environment, including your credentials.
     *
     * For information, see this documentation topic:
     *
     * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.
     * html
     */

    public static void main(String[] args) {
        final String usage = ""

            Usage:
            <functionName>\s

            Where:
            functionName - The name of the Lambda function\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        Region region = Region.US_WEST_2;
        LambdaClient awsLambda = LambdaClient.builder()
            .region(region)
            .build();

        invokeFunction(awsLambda, functionName);
    }
}
```

```
        awsLambda.close();
    }

    public static void invokeFunction(LambdaClient awsLambda, String
functionName) {

        InvokeResponse res = null;
        try {
            // Need a SdkBytes instance for the payload.
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("inputValue", "2000");
            String json = jsonObj.toString();
            SdkBytes payload = SdkBytes.fromUtf8String(json);

            // Setup an InvokeRequest.
            InvokeRequest request = InvokeRequest.builder()
                .functionName(functionName)
                .payload(payload)
                .build();

            res = awsLambda.invoke(request);
            String value = res.payload().asUtf8String();
            System.out.println(value);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for Java 2.x.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for JavaScript.

## Kotlin

### SDK para Kotlin

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
suspend fun invokeFunction(functionNameVal: String) {
  val json = """"{"inputValue":"1000}"""""
```

```

val byteArray = json.trimIndent().encodeToByteArray()
val request =
    InvokeRequest {
        functionName = functionNameVal
        logType = LogType.Tail
        payload = byteArray
    }

LambdaClient { region = "us-west-2" }.use { awsLambda ->
    val res = awsLambda.invoke(request)
    println("${res.payload?.toString(Charsets.UTF_8)}")
    println("The log result is ${res.logResult}")
}
}

```

- Para obtener información sobre la API, consulte [Invocar](#) en la Referencia de la API de AWS SDK para Kotlin.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

public function invoke($functionName, $params, $logType = 'None')
{
    return $this->lambdaClient->invoke([
        'FunctionName' => $functionName,
        'Payload' => json_encode($params),
        'LogType' => $logType,
    ]);
}

```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for PHP.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
        dict
                               is serialized to JSON before it is sent to
        Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
        included in
                               the response.
        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
        return response
```

- Para obtener información sobre la API, consulte [Invocar](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Invokes a Lambda function.
  # @param function_name [String] The name of the function to invoke.
  # @param payload [nil] Payload containing runtime parameters.
  # @return [Object] The response from the function invocation.
  def invoke_function(function_name, payload = nil)
    params = { function_name: function_name }
    params[:payload] = payload unless payload.nil?
    @lambda_client.invoke(params)
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error executing #{function_name}:\n
    #{e.message}")
  end
end
```

- Para obtener información sobre la API, consulte [Invocar](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/** Invoke the lambda function using calculator InvokeArgs. */
pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
    info!(?args, "Invoking {}", self.lambda_name);
    let payload = serde_json::to_string(&args)?;
    debug!(?payload, "Sending payload");
    self.lambda_client
        .invoke()
        .function_name(self.lambda_name.clone())
        .payload(Blob::new(payload))
        .send()
        .await
        .map_err(anyhow::Error::from)
}

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
```

- Para obtener información sobre la API, consulte [Invocar](#) en la Referencia de la API de AWS SDK para Rust.

## SAP ABAP

## SDK de SAP ABAP

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

TRY.
  DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
    `{` &&
    ` "action": "increment",` &&
    ` "number": 10` &&
    `}`
  ).
  oo_result = lo_lmd->invoke(
    " oo_result is returned for
testing purposes. "
    iv_functionname = iv_function_name
    iv_payload = lv_json
  ).
  MESSAGE 'Lambda function invoked.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
  MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdinvrequestcontex.
  MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
CATCH /aws1/cx_lmdinvalidzipfileex.
  MESSAGE 'The deployment package could not be unzipped.' TYPE 'E'.
CATCH /aws1/cx_lmdrequesttoolargeex.
  MESSAGE 'Invoke request body JSON input limit was exceeded by the request
payload.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
  MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
  MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
  MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
  MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.

```



```
CATCH /aws1/cx_lmdunsuppedmediatyp00.  
    MESSAGE 'Invoke request body does not have JSON as its content type.'  
TYPE 'E'.  
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [Invoke](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **ListFunctions** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar ListFunctions.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>  
/// Get a list of Lambda functions.  
/// </summary>  
/// <returns>A list of FunctionConfiguration objects.</returns>  
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()  
{  
    var functionList = new List<FunctionConfiguration>();
```

```
var functionPaginator =
    _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
await foreach (var function in functionPaginator.Functions)
{
    functionList.Add(function);
}

return functionList;
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for .NET.

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

std::vector<Aws::String> functions;
Aws::String marker;

do {
    Aws::Lambda::Model::ListFunctionsRequest request;
    if (!marker.empty()) {
        request.SetMarker(marker);
    }
}
```

```

    Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
        request);

    if (outcome.IsSuccess()) {
        const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
        std::cout << result.GetFunctions().size()
            << " lambda functions were retrieved." << std::endl;

        for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
            functions.push_back(functionConfiguration.GetFunctionName());
            std::cout << functions.size() << " "
                << functionConfiguration.GetDescription() << std::endl;
            std::cout << " "
                <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                functionConfiguration.GetRuntime()) << ": "
                << functionConfiguration.GetHandler()
                << std::endl;
        }
        marker = result.GetNextMarker();
    }
    else {
        std::cerr << "Error with Lambda::ListFunctions. "
            << outcome.GetError().GetMessage()
            << std::endl;
    }
} while (!marker.empty());

```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

#### Cómo recuperar una lista de funciones de Lambda

En el siguiente ejemplo de `list-functions`, se muestra una lista de todas las funciones para el usuario actual:

**aws lambda list-functions**

Salida:

```
{
  "Functions": [
    {
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "Version": "$LATEST",
      "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbl/RMr0fT/I=",
      "FunctionName": "helloworld",
      "MemorySize": 128,
      "RevisionId": "1718e831-badf-4253-9518-d0644210af7b",
      "CodeSize": 294,
      "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:helloworld",
      "Handler": "helloworld.handler",
      "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-
role-zgur6bf4",
      "Timeout": 3,
      "LastModified": "2023-09-23T18:32:33.857+0000",
      "Runtime": "nodejs18.x",
      "Description": ""
    },
    {
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "Version": "$LATEST",
      "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVrmY6E=",
      "FunctionName": "my-function",
      "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
      },
      "MemorySize": 256,
      "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
      "CodeSize": 266,
      "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",

```

```

        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
        "Timeout": 3,
        "LastModified": "2023-10-01T16:47:28.490+0000",
        "Runtime": "nodejs18.x",
        "Description": ""
    },
    {
        "Layers": [
            {
                "CodeSize": 41784542,
                "Arn": "arn:aws:lambda:us-
west-2:420165488524:layer:AWSLambda-Python37-SciPy1x:2"
            },
            {
                "CodeSize": 4121,
                "Arn": "arn:aws:lambda:us-
west-2:123456789012:layer:pythonLayer:1"
            }
        ],
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "$LATEST",
        "CodeSha256": "ZQukCqxtkqFgyF2cU41Avj99TKQ/hNihPtDtRcc08mI=",
        "FunctionName": "my-python-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 128,
        "RevisionId": "80b4eabc-acf7-4ea8-919a-e874c213707d",
        "CodeSize": 299,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
python-function",
        "Handler": "lambda_function.lambda_handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/my-python-
function-role-z5g7dr6n",
        "Timeout": 3,
        "LastModified": "2023-10-01T19:40:41.643+0000",
        "Runtime": "python3.11",
        "Description": ""
    }

```

```

    }
  ]
}

```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información sobre la API, consulte [ListFunctions](#) en la Referencia de comandos del AWS CLI.

## Go

### SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// ListFunctions lists up to maxItems functions for the account. This function
// uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(maxItems int)
[]types.FunctionConfiguration {
    var functions []types.FunctionConfiguration
    paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
    &lambda.ListFunctionsInput{
        MaxItems: aws.Int32(int32(maxItems)),
    })
    for paginator.HasMorePages() && len(functions) < maxItems {
        pageOutput, err := paginator.NextPage(context.TODO())
    }
}

```

```
if err != nil {
    log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
}
functions = append(functions, pageOutput.Functions...)
}
return functions
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for Go.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const listFunctions = () => {
    const client = new LambdaClient({});
    const command = new ListFunctionsCommand({});

    return client.send(command);
};
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for JavaScript.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function listFunctions($maxItems = 50, $marker = null)
{
    if (is_null($marker)) {
        return $this->lambdaClient->listFunctions([
            'MaxItems' => $maxItems,
        ]);
    }

    return $this->lambdaClient->listFunctions([
        'Marker' => $marker,
        'MaxItems' => $maxItems,
    ]);
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for PHP.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se muestran todas las funciones de Lambda con un tamaño de código ordenado

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
RunTime, Timeout, CodeSize
```

Salida:



FunctionName	Runtime	Timeout
CodeSize		
-----	-----	-----
-----		
test	python2.7	3
243		
MyLambdaFunction123	python3.8	600
659		
myfuncpython1	python3.8	303
675		

- Para obtener información sobre la API, consulte [ListFunctions](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
```

```
        print(f"\t{desc}")
        print(f"\t{func['Runtime']}: {func['Handler']}")
except ClientError as err:
    logger.error(
        "Couldn't list functions. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response["functions"].each do |function|
        functions.append(function["function_name"])
      end
    end
  end
end
```

```
end
functions
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
end
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/** List all Lambda functions in the current Region. */
pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
  info!("Listing lambda functions");
  self.lambda_client
    .list_functions()
    .send()
    .await
    .map_err(anyhow::Error::from)
}
```

- Para obtener información sobre la API, consulte [ListFunctions](#) en la Referencia de la API de AWS SDK para Rust.

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
TRY.
    oo_result = lo_lmd->listfunctions( ).      " oo_result is returned for
testing purposes. "
    DATA(lt_functions) = oo_result->get_functions( ).
    MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [ListFunctions](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **ListProvisionedConcurrencyConfigs** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListProvisionedConcurrencyConfigs`.

## CLI

## AWS CLI

Obtención de una lista de configuraciones de simultaneidad aprovisionada

En el siguiente ejemplo `list-provisioned-concurrency-configs`, se enumeran las configuraciones de simultaneidad aprovisionadas para la función especificada.

```
aws lambda list-provisioned-concurrency-configs \  
  --function-name my-function
```

Salida:

```
{  
  "ProvisionedConcurrencyConfigs": [  
    {  
      "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-  
function:GREEN",  
      "RequestedProvisionedConcurrentExecutions": 100,  
      "AvailableProvisionedConcurrentExecutions": 100,  
      "AllocatedProvisionedConcurrentExecutions": 100,  
      "Status": "READY",  
      "LastModified": "2019-12-31T20:29:00+0000"  
    },  
    {  
      "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-  
function:BLUE",  
      "RequestedProvisionedConcurrentExecutions": 100,  
      "AvailableProvisionedConcurrentExecutions": 100,  
      "AllocatedProvisionedConcurrentExecutions": 100,  
      "Status": "READY",  
      "LastModified": "2019-12-31T20:28:49+0000"  
    }  
  ]  
}
```

- Para obtener información sobre la API, consulte [ListProvisionedConcurrencyConfigs](#) en la Referencia del comando de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo obtiene la configuración de simultaneidad aprovisionada para una función de Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MyLambdaFunction123"
```

- Para obtener información sobre la API, consulte [ListProvisionedConcurrencyConfigs](#) en la Referencia de Cmdlets de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **ListTags** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListTags`.

### CLI

#### AWS CLI

Recuperación de la lista de etiquetas para una función de Lambda

En el siguiente ejemplo de `list-tags`, se muestran las etiquetas asociadas a la función de Lambda `my-function`.

```
aws lambda list-tags \  
  --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Salida:

```
{  
  "Tags": {  
    "Category": "Web Tools",  
    "Department": "Sales"  
  }  
}
```

Para obtener más información, consulte [Etiquetas de funciones de Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [ListTags](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: recupera las etiquetas y sus valores actualmente configurados en la función especificada.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

Salida:

```
Key      Value
---      -
California Sacramento
Oregon    Salem
Washington Olympia
```

- Para obtener información sobre la API, consulte [ListTags](#) en la Referencia de cmdlets de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **ListVersionsByFunction** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListVersionsByFunction`.

### CLI

#### AWS CLI

Recuperación de la lista de versiones de una función

En el siguiente ejemplo de `list-versions-by-function`, se muestra la lista de versiones de la función de Lambda `my-function`.

```
aws lambda list-versions-by-function \  
  --function-name my-function
```

Salida:

```
{  
  "Versions": [  
    {  
      "TracingConfig": {  
        "Mode": "PassThrough"  
      },  
      "Version": "$LATEST",  
      "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVmY6E=",  
      "FunctionName": "my-function",  
      "VpcConfig": {  
        "SubnetIds": [],  
        "VpcId": "",  
        "SecurityGroupIds": []  
      },  
      "MemorySize": 256,  
      "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",  
      "CodeSize": 266,  
      "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:$LATEST",  
      "Handler": "index.handler",  
      "Role": "arn:aws:iam:123456789012:role/service-role/  
helloWorldPython-role-uy3l9qqq",  
      "Timeout": 3,  
      "LastModified": "2019-10-01T16:47:28.490+0000",  
      "Runtime": "nodejs10.x",  
      "Description": ""  
    },  
    {  
      "TracingConfig": {  
        "Mode": "PassThrough"  
      },  
      "Version": "1",  
      "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJm1KidWaaCgk=",  
      "FunctionName": "my-function",  
      "VpcConfig": {
```



```

        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "MemorySize": 256,
    "RevisionId": "949c8914-012e-4795-998c-e467121951b1",
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:1",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
    "Timeout": 3,
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "Runtime": "nodejs10.x",
    "Description": "new version"
},
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "Version": "2",
    "CodeSha256": "sU0cJ2/h0ZevwV/1TxCuQqK3gDZP3i8gUoqUUVmY6E=",
    "FunctionName": "my-function",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "MemorySize": 256,
    "RevisionId": "cd669f21-0f3d-4e1c-9566-948837f2e2ea",
    "CodeSize": 266,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
    "Handler": "index.handler",
    "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qq",
    "Timeout": 3,
    "LastModified": "2019-10-01T16:47:28.490+0000",
    "Runtime": "nodejs10.x",
    "Description": "newer version"
}
]

```

```
}

```

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [ListVersionsByFunction](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo devuelve la lista de configuraciones específicas de la versión para cada versión de la función de Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Salida:

FunctionName	Runtime	MemorySize	Timeout	CodeSize	LastModified
RoleName					
-----	-----	-----	-----	-----	-----
-----					
MylambdaFunction123	python3.8	128	600	659	
2020-01-10T03:20:56.390+0000	lambda				
MylambdaFunction123	python3.8	128	5	1426	
2019-12-25T09:19:02.238+0000	lambda				
MylambdaFunction123	python3.8	128	5	1426	
2019-12-25T09:39:36.779+0000	lambda				
MylambdaFunction123	python3.8	128	600	1426	
2019-12-25T09:52:59.872+0000	lambda				

- Para obtener información sobre la API, consulte [ListVersionsByFunction](#) en la Referencia de cmdlets de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **PublishVersion** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar PublishVersion.

### CLI

#### AWS CLI

Publicación de una nueva versión de la función de Lambda

En el siguiente ejemplo de `publish-version`, se publica una nueva versión de la función de Lambda `my-function`.

```
aws lambda publish-version \  
  --function-name my-function
```

Salida:

```
{  
  "TracingConfig": {  
    "Mode": "PassThrough"  
  },  
  "CodeSha256": "dBG9m8SGdm1Ejw/JYX1hhvCrAv5TxvXsbl/RMr0fT/I=",  
  "FunctionName": "my-function",  
  "CodeSize": 294,  
  "RevisionId": "f31d3d39-cc63-4520-97d4-43cd44c94c20",  
  "MemorySize": 128,  
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:3",  
  "Version": "2",  
  "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-  
zгур6bf4",  
  "Timeout": 3,  
  "LastModified": "2019-09-23T18:32:33.857+0000",  
  "Handler": "my-function.handler",  
  "Runtime": "nodejs10.x",  
  "Description": ""  
}
```

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener detalles de la API, consulte [PublishVersion](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo crea una versión para la instantánea existente del código de función de Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing Existing Snapshot of function code as a new version through Powershell"
```

- Para obtener detalles de la API, consulte [PublishVersion](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **PutFunctionConcurrency** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `PutFunctionConcurrency`.

### CLI

#### AWS CLI

Configuración de un límite de simultaneidad reservado para una función

En el siguiente ejemplo de `put-function-concurrency`, se configuran 100 ejecuciones simultáneas reservadas para la función `my-function`.

```
aws lambda put-function-concurrency \  
  --function-name my-function \  
  --reserved-concurrent-executions 100
```

Salida:

```
{
```

```
"ReservedConcurrentExecutions": 100
}
```

Para obtener más información, consulte [Reserva de simultaneidad para una función de Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [PutFunctionConcurrency](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se aplica la configuración de simultaneidad de la función en su conjunto.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Para obtener información sobre la API, consulte [PutFunctionConcurrency](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **PutProvisionedConcurrencyConfig** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `PutProvisionedConcurrencyConfig`.

### CLI

#### AWS CLI

#### Asignación de simultaneidad aprovisionada

En el siguiente ejemplo de `put-provisioned-concurrency-config`, se asignan 100 simultaneidades aprovisionadas para el alias BLUE de la función especificada.

```
aws lambda put-provisioned-concurrency-config \  
  --function-name my-function \  
  --qualifier BLUE \  
  --provisioned-concurrent-executions 100
```

Salida:

```
{  
  "Requested ProvisionedConcurrentExecutions": 100,  
  "Allocated ProvisionedConcurrentExecutions": 0,  
  "Status": "IN_PROGRESS",  
  "LastModified": "2019-11-21T19:32:12+0000"  
}
```

- Para obtener información sobre la API, consulte [PutProvisionedConcurrencyConfig](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: en este ejemplo se agrega una configuración de simultaneidad aprovisionada al alias de una función

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MyLambdaFunction123" -  
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Para obtener información sobre la API, consulte [PutProvisionedConcurrencyConfig](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **RemovePermission** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `RemovePermission`.

## CLI

### AWS CLI

Eliminación de permisos de una función de Lambda existente

En el siguiente ejemplo de `remove-permission`, se elimina el permiso para invocar una función denominada `my-function`.

```
aws lambda remove-permission \  
  --function-name my-function \  
  --statement-id sns
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Uso de políticas basadas en recursos para AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [RemovePermission](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo elimina la política de función para el `StatementID` especificado de una función de Lambda.

```
$policy = Get-LMPolicy -FunctionName "MyLambdaFunction123" -Select Policy |  
  ConvertFrom-Json | Select-Object -ExpandProperty Statement  
Remove-LMPPermission -FunctionName "MyLambdaFunction123" -StatementId  
  $policy[0].Sid
```

- Para obtener información sobre la API, consulte [RemovePermission](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **TagResource** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `TagResource`.

### CLI

#### AWS CLI

Adición de etiquetas a una función de Lambda existente

En el siguiente ejemplo de `tag-resource`, se agrega una etiqueta con el nombre de clave `DEPARTMENT` y un valor de `Department A` a la función de Lambda especificada.

```
aws lambda tag-resource \  
  --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \  
  --tags "DEPARTMENT=Department A"
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Etiquetas de funciones de Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para ver los detalles de la API, consulte [TagResource](#) en la Referencia del comando de la AWS CLI.

### PowerShell

#### Herramientas para PowerShell

Ejemplo 1: agrega las tres etiquetas (Washington, Oregon y California) y sus valores asociados a la función especificada identificada por su ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-  
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";  
  "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Para obtener información sobre la API, consulte [TagResource](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.



Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **UntagResource** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar UntagResource.

### CLI

#### AWS CLI

Eliminación de etiquetas de una función de Lambda existente

En el siguiente ejemplo de `untag-resource`, se elimina la etiqueta con el nombre de clave DEPARTMENT de la función de Lambda `my-function`.

```
aws lambda untag-resource \  
  --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \  
  --tag-keys DEPARTMENT
```

Este comando no genera ninguna salida.

Para obtener más información, consulte [Etiquetas de funciones de Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [UntagResource](#) en la Referencia de comandos de la AWS CLI.

### PowerShell

#### Herramientas para PowerShell

Ejemplo 1: elimina las etiquetas suministradas de una función. El cmdlet solicitará confirmación antes de continuar, a menos que se especifique el modificador `-Force`. Se realiza una sola llamada al servicio para eliminar las etiquetas.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-  
west-2:123456789012:function:MyFunction" -TagKey  
  "Washington","Oregon","California"
```

Ejemplo 2: elimina las etiquetas suministradas de una función. El cmdlet solicitará confirmación antes de continuar, a menos que se especifique el modificador `-Force`. Una vez realizada la llamada al servicio por etiqueta suministrada.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource  
"arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Para obtener información sobre la API, consulte [UntagResource](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **UpdateAlias** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `UpdateAlias`.

### CLI

#### AWS CLI

Actualización del alias de una función

En el siguiente ejemplo de `update-alias`, se actualiza el alias nombrado `LIVE` para que apunte a la versión 3 de la función de Lambda `my-function`.

```
aws lambda update-alias \  
  --function-name my-function \  
  --function-version 3 \  
  --name LIVE
```

Salida:

```
{  
  "FunctionVersion": "3",  
  "Name": "LIVE",  
  "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-  
function:LIVE",
```

```
"RevisionId": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",  
"Description": "alias for live version of function"  
}
```

Para obtener más información, consulte [Configuración de los alias de las funciones de AWS Lambda](#) en la Guía para desarrolladores de AWS Lambda.

- Para obtener información sobre la API, consulte [UpdateAlias](#) en la Referencia de comandos de la AWS CLI.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo actualiza la configuración de un alias de función de Lambda existente. Actualiza el valor de RoutingConfiguration para transferir el 60 % (0,6) del tráfico a la versión 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description  
" Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -  
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6"}
```

- Para obtener información sobre la API, consulte [UpdateAlias](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de **UpdateFunctionCode** con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar UpdateFunctionCode.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

## .NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{
    var functionCodeRequest = new UpdateFunctionCodeRequest
    {
        FunctionName = functionName,
        Publish = true,
        S3Bucket = bucketName,
        S3Key = key,
    };

    var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
    Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for .NET.

## C++

## SDK para C++

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

    Aws::Lambda::Model::UpdateFunctionCodeRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                          std::ios_base::in | std::ios_base::binary);
    if (!ifstream.is_open()) {
        std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;
    }

#ifdef USE_CPP_LAMBDA_FUNCTION
    std::cerr
        << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
        << std::endl;
#endif

    deleteLambdaFunction(client);
    deleteIamRole(clientConfig);
    return false;
}

    Aws::StringStream buffer;
    buffer << ifstream.rdbuf();
    request.SetZipFile(
        Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                               buffer.str().length()));

    request.SetPublish(true);
```

```
    Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda code was successfully updated." <<
std::endl;
    }
    else {
        std::cerr << "Error with Lambda::UpdateFunctionCode. "
        << outcome.GetError().GetMessage()
        << std::endl;
    }
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

#### Cómo actualizar el código de una función de Lambda

En el siguiente ejemplo de `update-function-code`, se reemplaza el código de la versión no publicada (`$LATEST`) de la función `my-function` por el contenido del archivo zip especificado.

```
aws lambda update-function-code \
  --function-name my-function \
  --zip-file fileb://my-function.zip
```

Salida:

```
{
  "FunctionName": "my-function",
  "LastModified": "2019-09-26T20:28:40.438+0000",
  "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
  "MemorySize": 256,
  "Version": "$LATEST",
```

```

    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
    uy3l9qq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Handler": "index.handler"
}

```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información acerca de la API, consulte [UpdateFunctionCode](#) en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

```

```
// UpdateFunctionCode updates the code for the Lambda function specified by
// functionName.
// The existing code for the Lambda function is entirely replaced by the code in
// the
// zipPackage buffer. After the update action is called, a
// lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(functionName string, zipPackage
*bytes.Buffer) types.State {
    var state types.State
    _, err := wrapper.LambdaClient.UpdateFunctionCode(context.TODO(),
&lambda.UpdateFunctionCodeInput{
    FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
})
    if err != nil {
        log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
functionName, err)
    } else {
        waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
        funcOutput, err := waiter.WaitForOutput(context.TODO(),
&lambda.GetFunctionInput{
        FunctionName: aws.String(functionName)}, 1*time.Minute)
        if err != nil {
            log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
        } else {
            state = funcOutput.Configuration.State
        }
    }
    return state
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for Go.



## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for JavaScript.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
```

```
{
    return $this->lambdaClient->updateFunctionCode([
        'FunctionName' => $functionName,
        'S3Bucket' => $s3Bucket,
        'S3Key' => $s3Key,
    ]);
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for PHP.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: actualiza la función denominada “MyFunction” con contenido nuevo contenido en el archivo zip especificado. Para una función de Lambda C# .NET Core, el archivo zip debe contener el ensamblaje compilado.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Ejemplo 2: este ejemplo es similar al anterior, pero utiliza un objeto de Amazon S3 que contiene el código actualizado para actualizar la función.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName mybucket -Key
UpdatedCode.zip
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
        contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
        in
                                .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the code for a Lambda function by submitting a .zip archive that
  # contains
  # the code for the function.

  # @param function_name: The name of the function to update.
  # @param deployment_package: The function code to update, packaged as bytes in
  #                               .zip format.
  # @return: Data about the update, including the status.
  def update_function_code(function_name, deployment_package)
    @lambda_client.update_function_code(
      function_name: function_name,
      zip_file: deployment_package
    )
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name}) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
    rescue Aws::Lambda::Errors::ServiceException => e
      @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
      nil
    rescue Aws::Waiters::Errors::WaiterFailed => e
```

```
@logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
end
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
pub async fn update_function_code(
    &self,
    zip_file: PathBuf,
    key: String,
) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
    let function_code = self.prepare_function(zip_file, Some(key)).await?;

    info!("Updating code for {}", self.lambda_name);
    let update = self
        .lambda_client
        .update_function_code()
        .function_name(self.lambda_name.clone())
        .s3_bucket(self.bucket.clone())
        .s3_key(function_code.s3_key().unwrap().to_string())
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(update)
```

```
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));


        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionCode](#) en la Referencia de la API del AWS SDK para Rust.

## SAP ABAP

## SDK de SAP ABAP

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
TRY.
    oo_result = lo_lmd->updatefunctioncode(      " oo_result is returned for
testing purposes. "
        iv_functionname = iv_function_name
        iv_zipfile = io_zip_file
    ).

    MESSAGE 'Lambda function code updated.' TYPE 'I'.
CATCH /aws1/cx_lmdcodesigningcfn00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdcodestorageexc00.
    MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
CATCH /aws1/cx_lmdinvalidcodesigex.
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdresourceconflictex.
    MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdserviceexception.
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
CATCH /aws1/cx_lmdtoomanyrequestsex.
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [UpdateFunctionCode](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de `UpdateFunctionConfiguration` con un AWS SDK o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `UpdateFunctionConfiguration`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- [Primeros pasos con las funciones](#)

### .NET

#### AWS SDK for .NET

##### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/// <summary>
/// Update the code of a Lambda function.
/// </summary>
/// <param name="functionName">The name of the function to update.</param>
/// <param name="functionHandler">The code that performs the function's
actions.</param>
/// <param name="environmentVariables">A dictionary of environment
variables.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> UpdateFunctionConfigurationAsync(
    string functionName,
    string functionHandler,
    Dictionary<string, string> environmentVariables)
{
```



```
var request = new UpdateFunctionConfigurationRequest
{
    Handler = functionHandler,
    FunctionName = functionName,
    Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
};

var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

Console.WriteLine(response.LastModified);

return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for .NET.

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Lambda::LambdaClient client(clientConfig);

Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
request.SetFunctionName(LAMBDA_NAME);
Aws::Lambda::Model::Environment environment;
```

```
environment.AddVariables("LOG_LEVEL", "DEBUG");
request.SetEnvironment(environment);

Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
    request);

if (outcome.IsSuccess()) {
    std::cout << "The lambda configuration was successfully updated."
              << std::endl;
    break;
}

else {
    std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
              << outcome.GetError().GetMessage()
              << std::endl;
}
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for C++.

## CLI

### AWS CLI

Para modificar la configuración de una función

En el siguiente ejemplo de `update-function-configuration`, se modifica el tamaño de la memoria para que sea de 256 MB para la versión no publicada (\$LATEST) de la función `my-function`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --memory-size 256
```

Salida:

```
{
  "FunctionName": "my-function",
```

```
"LastModified": "2019-09-26T20:28:40.438+0000",
"RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
"MemorySize": 256,
"Version": "$LATEST",
"Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-uy3l9yq",
"Timeout": 3,
"Runtime": "nodejs10.x",
"TracingConfig": {
  "Mode": "PassThrough"
},
"CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
"Description": "",
"VpcConfig": {
  "SubnetIds": [],
  "VpcId": "",
  "SecurityGroupIds": []
},
"CodeSize": 304,
"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
"Handler": "index.handler"
}
```

Para obtener más información, consulte [Configuración las funciones de Lambda de AWS](#) en la Guía para desarrolladores de Lambda de AWS.

- Para obtener información acerca de la API, consulte [UpdateFunctionConfiguration](#) en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
// FunctionWrapper encapsulates function actions used in the examples.
```

```
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// UpdateFunctionConfiguration updates a map of environment variables configured
// for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(functionName string,
    envVars map[string]string) {
    _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(context.TODO(),
        &lambda.UpdateFunctionConfigurationInput{
            FunctionName: aws.String(functionName),
            Environment: &types.Environment{Variables: envVars},
        })
    if err != nil {
        log.Panicf("Couldn't update configuration for %v. Here's why: %v",
            functionName, err)
    }
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for Go.

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
const updateFunctionConfiguration = (funcName) => {
    const client = new LambdaClient({});
    const config = readFileSync(`${dirname}../functions/config.json`).toString();
```

```
const command = new UpdateFunctionConfigurationCommand({
  ...JSON.parse(config),
  FunctionName: funcName,
});
return client.send(command);
};
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for JavaScript.

## PHP

### SDK para PHP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
{
    return $this->lambdaClient->updateFunctionConfiguration([
        'FunctionName' => $functionName,
        'Handler' => "$handler.lambda_handler",
        'Environment' => $environment,
    ]);
}
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for PHP.

## PowerShell

### Herramientas para PowerShell

Ejemplo 1: este ejemplo actualiza la configuración de la función de Lambda existente

```
Update-LMFunctionConfiguration -FunctionName "MyLambdaFunction123" -Handler
"lambda_function.launch_instance" -Timeout 600 -Environment_Variable
@{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
123456789101:MyfirstTopic
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la Referencia de Cmdlet de AWS Tools for PowerShell.

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function configuration %s. Here's why: %s: %s",
                function_name,
```

```
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response
```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la Referencia de la API de AWS SDK para Python (Boto3).

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
class LambdaWrapper
  attr_accessor :lambda_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the environment variables for a Lambda function.
  # @param function_name: The name of the function to update.
  # @param log_level: The log level of the function.
  # @return: Data about the update, including the status.
  def update_function_configuration(function_name, log_level)
    @lambda_client.update_function_configuration({
      function_name: function_name,
      environment: {
        variables: {
          "LOG_LEVEL" => log_level
        }
      }
    })
  end
end
```

```

    }
  }
})

@lambda_client.wait_until(:function_updated_v2, { function_name:
function_name}) do |w|
  w.max_attempts = 5
  w.delay = 5
end
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
rescue Aws::Waiters::Errors::WaiterFailed => e
  @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
end

```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la referencia de la API de AWS SDK for Ruby.

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

/** Update the environment for a function. */
pub async fn update_function_configuration(
    &self,
    environment: Environment,
) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
    info!(
        ?environment,
        "Updating environment for {}", self.lambda_name
    );
    let updated = self
        .lambda_client

```



```

        .update_function_configuration()
        .function_name(self.lambda_name.clone())
        .environment(environment)
        .send()
        .await
        .map_err(anyhow::Error::from)?;

    self.wait_for_function_ready().await?;

    Ok(updated)
}

```

- Para obtener información sobre la API, consulte [UpdateFunctionConfiguration](#) en la Referencia de la API del AWS SDK para Rust.

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```

TRY.
    oo_result = lo_lmd->updatefunctionconfiguration(      " oo_result is
returned for testing purposes. "
        iv_functionname = iv_function_name
        iv_runtime = iv_runtime
        iv_description = 'Updated Lambda function'
        iv_memorysize = iv_memory_size
    ).

    MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
CATCH /aws1/cx_lmdcodesigningcfgno00.
    MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdcodeverification00.
    MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.

```

```
CATCH /aws1/cx_lmdinvalidcodesigex.  
    MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.  
CATCH /aws1/cx_lmdinvparamvalueex.  
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.  
CATCH /aws1/cx_lmdresourceconflictex.  
    MESSAGE 'Resource already exists or another operation is in progress.'  
TYPE 'E'.  
CATCH /aws1/cx_lmdresourcenotfoundex.  
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.  
CATCH /aws1/cx_lmdserviceexception.  
    MESSAGE 'An internal problem was encountered by the AWS Lambda service.'  
TYPE 'E'.  
CATCH /aws1/cx_lmdtoomanyrequestsex.  
    MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.  
ENDTRY.
```

- Para obtener detalles sobre la API, consulte [UpdateFunctionConfiguration](#) en la Referencia de la API del AWS SDK para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Escenarios de Lambda con SDK de AWS

En los siguientes ejemplos de código se muestra cómo implementar escenarios comunes en Lambda con el SDK AWS. Estos escenarios muestran cómo llevar a cabo tareas específicas llamando a varias funciones dentro de Lambda. En cada escenario se incluye un enlace a GitHub, con instrucciones de configuración y ejecución del código.

### Ejemplos

- [Confirmación de manera automática a los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK](#)
- [Migración en forma automática los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK](#)
- [Primeros pasos para la creación y la invocación de funciones de Lambda con un AWS SDK](#)
- [Escriba datos de actividad personalizados con una función de Lambda tras la autenticación de usuarios de Amazon Cognito mediante un AWS SDK](#)

## Confirmación de manera automática a los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK

En el siguiente ejemplo de código, se muestra cómo confirmar de manera automática los usuarios conocidos de Amazon Cognito con una función de Lambda.

- Configure un grupo de usuarios para que llame a una función de Lambda para el desencadenador PreSignUp.
- Inscripción de un usuario mediante Amazon Cognito
- La función de Lambda escanea una tabla de DynamoDB y confirma de manera automática los usuarios conocidos.
- Inicie sesión con el nuevo usuario y, luego, elimine los recursos.

Go

SDK para Go V2

### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
// AutoConfirm separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type AutoConfirm struct {
    helper          IScenarioHelper
    questioner     demotools.IQuestioner
    resources       Resources
    cognitoActor   *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) AutoConfirm {
```

```
scenario := AutoConfirm{
    helper:      helper,
    questioner:  questioner,
    resources:   Resources{},
    cognitoActor: &actions.CognitoActions{CognitoClient:
cognitoidentityprovider.NewFromConfig(sdkConfig)},
}
scenario.resources.init(scenario.cognitoActor, questioner)
return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(userPoolId string, functionArn
string) {
    log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
Cognito.\n" +
        "This trigger happens when a user signs up, and lets your function take action
before the main Cognito\n" +
        "sign up processing occurs.\n")
    err := runner.cognitoActor.UpdateTriggers(
        userPoolId,
        actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
aws.String(functionArn)})
    if err != nil {
        panic(err)
    }
    log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
trigger.\n",
        functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you
specify.
func (runner *AutoConfirm) SignUpUser(clientId string, usersTable string)
(string, string) {
    log.Println("Let's sign up a user to your Cognito user pool. When the user's
email matches an email in the\n" +
        "DynamoDB known users table, it is automatically verified and the user is
confirmed.")

    knownUsers, err := runner.helper.GetKnownUsers(usersTable)
    if err != nil {
        panic(err)
    }
}
```

```

}
userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
knownUsers.UserNameList())
user := knownUsers.Users[userChoice]

var signedUp bool
var userConfirmed bool
password := runner.questioner.AskPassword("Enter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
for !signedUp {
log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
user.UserEmail)
userConfirmed, err = runner.cognitoActor.SignUp(clientId, user.UserName,
password, user.UserEmail)
if err != nil {
var invalidPassword *types.InvalidPasswordException
if errors.As(err, &invalidPassword) {
password = runner.questioner.AskPassword("Enter another password:", 8)
} else {
panic(err)
}
} else {
signedUp = true
}
}
log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(clientId string, userName string, password
string) string {
runner.questioner.Ask("Press Enter when you're ready to continue.")
log.Printf("Let's sign in as %v...\n", userName)
authResult, err := runner.cognitoActor.SignIn(clientId, userName, password)
if err != nil {
panic(err)
}
log.Printf("Successfully signed in. Your access token starts with: %v...\n",
(*authResult.AccessToken)[:10])

```

```
log.Println(strings.Repeat("-", 88))
return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(stackName string) {
defer func() {
if r := recover(); r != nil {
log.Println("Something went wrong with the demo.")
runner.resources.Cleanup()
}
}()

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome\n")

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(stackName)
if err != nil {
panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(stackOutputs["TableName"])

runner.AddPreSignUpTrigger(stackOutputs["UserPoolId"],
stackOutputs["AutoConfirmFunctionArn"])
runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
userName, password := runner.SignUpUser(stackOutputs["UserPoolClientId"],
stackOutputs["TableName"])
runner.helper.ListRecentLogEvents(stackOutputs["AutoConfirmFunction"])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
runner.SignInUser(stackOutputs["UserPoolClientId"], userName, password))

runner.resources.Cleanup()

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}
```

## Controle el desencadenador PreSignUp con una función de Lambda.

```
const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    userEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := attributevalue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
error) {
    log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
event.UserName)
    if event.TriggerSource != "PreSignUp_SignUp" {
        // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
        // response from this handler.
        return event, nil
    }
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserEmail: event.Request.UserAttributes["email"],
    }
    log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
    output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
```

```
    Key:      user.GetKey(),
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Error looking up email %v.\n", user.UserEmail)
    return event, err
}
if output.Item == nil {
    log.Printf("Email %v not found. Email verification is required.\n",
user.UserEmail)
    return event, err
}

err = attributevalue.UnmarshalMap(output.Item, &user)
if err != nil {
    log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
    return event, err
}

if user.UserName != event.UserName {
    log.Printf("UserEmail %v found, but stored UserName '%v' does not match
supplied UserName '%v'. Verification is required.\n",
    user.UserEmail, user.UserName, event.UserName)
} else {
    log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
    event.Response.AutoConfirmUser = true
    event.Response.AutoVerifyEmail = true
}

return event, err
}

func main() {
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
}
```



Cree una estructura que lleve a cabo las tareas habituales.

```
// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(stackName string) (actions.StackOutputs, error)
    PopulateUserTable(tableName string)
    GetKnownUsers(tableName string) (actions.UserList, error)
    AddKnownUser(tableName string, user actions.User)
    ListRecentLogEvents(functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor *actions.CloudFormationActions
    cwActor *actions.CloudWatchLogsActions
    isTestRun bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
        dynamodb.NewFromConfig(sdkConfig)},
        cfnActor: &actions.CloudFormationActions{CfnClient:
        cloudformation.NewFromConfig(sdkConfig)},
        cwActor: &actions.CloudWatchLogsActions{CwlClient:
        cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
```

```
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
// structured format.
func (helper ScenarioHelper) GetStackOutputs(stackName string)
(actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(tableName string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(tableName)
    if err != nil {
        panic(err)
    }
}

// GetKnownUsers gets the users from the known users table in a structured
// format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
error) {
    knownUsers, err := helper.dynamoActor.Scan(tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(tableName string, user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(tableName, user)
    if err != nil {
        panic(err)
    }
}
```

```
// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
    your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
    *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(functionName,
    *logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
    for _, event := range events {
        log.Printf("\t\t%v", *event.Message)
    }
    log.Println(strings.Repeat("-", 88))
}
```

Cree una estructura que ajuste las acciones de Amazon Cognito.

```
type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
```

```
UserMigration
PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(userPoolId string,
    triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
        &cognitoidentityprovider.DescribeUserPoolInput{
            UserPoolId: aws.String(userPoolId),
        })
    if err != nil {
        log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
            userPoolId, err)
        return err
    }
    lambdaConfig := output.UserPool.LambdaConfig
    for _, trigger := range triggers {
        switch trigger.Trigger {
        case PreSignUp:
            lambdaConfig.PreSignUp = trigger.HandlerArn
        case UserMigration:
            lambdaConfig.UserMigration = trigger.HandlerArn
        case PostAuthentication:
            lambdaConfig.PostAuthentication = trigger.HandlerArn
        }
    }
    _, err = actor.CognitoClient.UpdateUserPool(context.TODO(),
        &cognitoidentityprovider.UpdateUserPoolInput{
            UserPoolId:    aws.String(userPoolId),
            LambdaConfig: lambdaConfig,
        })
    if err != nil {
        log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
    }
    return err
}
```

```
// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(clientId string, userName string, password
string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(context.TODO(),
    &cognitoidentityprovider.SignUpInput{
        ClientId: aws.String(clientId),
        Password: aws.String(password),
        Username: aws.String(userName),
        UserAttributes: []types.AttributeType{
            {Name: aws.String("email"), Value: aws.String(userEmail)},
        },
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
        }
    } else {
        confirmed = output.UserConfirmed
    }
    return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
authentication flow.
func (actor CognitoActions) SignIn(clientId string, userName string, password
string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
    &cognitoidentityprovider.InitiateAuthInput{
        AuthFlow:      "USER_PASSWORD_AUTH",
        ClientId:      aws.String(clientId),
        AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
    })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
```

```
    if errors.As(err, &resetRequired) {
        log.Println(*resetRequired.Message)
    } else {
        log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
    }
} else {
    authResult = output.AuthenticationResult
}
return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(clientId string, userName string)
(*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
&cognitoidentityprovider.ForgotPasswordInput{
    ClientId: aws.String(clientId),
    Username: aws.String(userName),
})
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(clientId string, code string,
userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
&cognitoidentityprovider.ConfirmForgotPasswordInput{
    ClientId:      aws.String(clientId),
    ConfirmationCode: aws.String(code),
    Password:      aws.String(password),
    Username:      aws.String(userName),
})
    if err != nil {
```

```
var invalidPassword *types.InvalidPasswordException
if errors.As(err, &invalidPassword) {
    log.Println(*invalidPassword.Message)
} else {
    log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
}
}
return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(userAccessToken string) error {
    _, err := actor.CognitoClient.DeleteUser(context.TODO(),
        &cognitoidentityprovider.DeleteUserInput{
            AccessToken: aws.String(userAccessToken),
        })
    if err != nil {
        log.Printf("Couldn't delete user. Here's why: %v\n", err)
    }
    return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(userPoolId string, userName string,
    userEmail string) error {
    _, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
        &cognitoidentityprovider.AdminCreateUserInput{
            UserPoolId:      aws.String(userPoolId),
            Username:       aws.String(userName),
            MessageAction: types.MessageActionTypeSuppress,
            UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
                aws.String(userEmail)}}},
        })
    if err != nil {
        var userExists *types.UsernameExistsException
        if errors.As(err, &userExists) {
            log.Printf("User %v already exists in the user pool.", userName)
            err = nil
        }
    }
}
```

```

    } else {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    }
}
return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(userPoolId string, userName
string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
&cognitoidentityprovider.AdminSetUserPasswordInput{
    Password:    aws.String(password),
    UserPoolId:  aws.String(userPoolId),
    Username:    aws.String(userName),
    Permanent:   true,
})
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
        }
    }
    return err
}

```

Cree una estructura que ajuste las acciones de DynamoDB.

```

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client

```



```
}

// User defines structured user data.
type User struct {
    UserName string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId string
    Time string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(tableName string) error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
    }
}
```

```
    writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}})
}
_, err = actor.DynamoClient.BatchWriteItem(context.TODO(),
&dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(tableName string) (UserList, error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(context.TODO(), &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(tableName string, user User) error {
    userItem, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
    }
    _, err = actor.DynamoClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
        Item:        userItem,
        TableName:   aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
    }
}
```

```

}
return err
}

```

Cree una estructura que ajuste las acciones de los registros de CloudWatch.

```

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
(types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
    &cloudwatchlogs.DescribeLogStreamsInput{
        Descending:    aws.Bool(true),
        Limit:         aws.Int32(1),
        LogGroupName:  aws.String(logGroupName),
        OrderBy:       types.OrderByLastEventTime,
    })
    if err != nil {
        log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
        logGroupName, err)
    } else {
        logStream = output.LogStreams[0]
    }
    return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
stream.
func (actor CloudWatchLogsActions) GetLogEvents(functionName string,
logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
    var events []types.OutputLogEvent
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.GetLogEvents(context.TODO(),
    &cloudwatchlogs.GetLogEventsInput{

```

```

    LogStreamName: aws.String(logStreamName),
    Limit:          aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
  })
  if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
      logStreamName, err)
  } else {
    events = output.Events
  }
  return events, err
}

```

Cree una estructura que ajuste las acciones de AWS CloudFormation.

```

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
  CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(stackName string) StackOutputs {
  output, err := actor.CfnClient.DescribeStacks(context.TODO(),
    &cloudformation.DescribeStacksInput{
      StackName: aws.String(stackName),
    })
  if err != nil || len(output.Stacks) == 0 {
    log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
      stackName, err)
  }
  stackOutputs := StackOutputs{}
  for _, out := range output.Stacks[0].Outputs {
    stackOutputs[*out.OutputKey] = *out.OutputValue
  }
  return stackOutputs
}

```

## Eliminación de recursos.

```
// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup() {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()

    wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
    "during this demo (y/n)?", "y")
    if wantDelete {
        for _, accessToken := range resources.userAccessTokens {
            err := resources.cognitoActor.DeleteUser(accessToken)
            if err != nil {
                log.Println("Couldn't delete user during cleanup.")
            }
        }
    }
}
```

```
panic(err)
}
log.Println("Deleted user.")
}
triggerList := make([]actions.TriggerInfo, len(resources.triggers))
for i := 0; i < len(resources.triggers); i++ {
    triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
}
err := resources.cognitoActor.UpdateTriggers(resources.userPoolId,
triggerList...)
if err != nil {
    log.Println("Couldn't update Cognito triggers during cleanup.")
    panic(err)
}
log.Println("Removed Cognito triggers from user pool.")
} else {
    log.Println("Be sure to remove resources when you're done with them to avoid
unexpected charges!")
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Go.
  - [DeleteUser](#)
  - [InitiateAuth](#)
  - [SignUp](#)
  - [UpdateUserPool](#)

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.


## Migración en forma automática los usuarios conocidos de Amazon Cognito con una función de Lambda mediante un AWS SDK

En el siguiente ejemplo de código, se muestra cómo migrar de manera automática los usuarios conocidos de Amazon Cognito con una función de Lambda.

- Configure un grupo de usuarios para que llame a una función de Lambda para el desencadenador `MigrateUser`.
- Inicie sesión en Amazon Cognito con un nombre de usuario y un correo electrónico que no estén en el grupo de usuarios.
- La función de Lambda escanea una tabla de DynamoDB y migra de manera automática los usuarios conocidos al grupo de usuarios.
- Realice el flujo en caso de olvido de contraseña para restablecer la contraseña respecto del usuario migrado.
- Inicie sesión como un nuevo usuario y, a continuación, elimine los recursos.

Go

SDK para Go V2

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
import (
    "errors"
    "fmt"
    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
```

```
type MigrateUser struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources   Resources
    cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) MigrateUser {
    scenario := MigrateUser{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
            cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(userPoolId string, functionArn
string) {
    log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
Cognito.\n" +
        "This trigger happens when an unknown user signs in, and lets your function
take action before Cognito\n" +
        "rejects the user.\n\n")
    err := runner.cognitoActor.UpdateTriggers(
        userPoolId,
        actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
            aws.String(functionArn)})
    if err != nil {
        panic(err)
    }
    log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
trigger.\n",
        functionArn, userPoolId)

    log.Println(strings.Repeat("-", 88))
}
```



```
// SignInUser adds a new user to the known users table and signs that user in to
// Amazon Cognito.
func (runner *MigrateUser) SignInUser(usersTable string, clientId string) (bool,
actions.User) {
    log.Println("Let's sign in a user to your Cognito user pool. When the username
    and email matches an entry in the\n" +
        "DynamoDB known users table, the email is automatically verified and the user
    is migrated to the Cognito user pool.")

    user := actions.User{}
    user.UserName = runner.questioner.Ask("\nEnter a username:")
    user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This
    email will be used to confirm user migration\n" +
        "during this example:")

    runner.helper.AddKnownUser(usersTable, user)

    var err error
    var resetRequired *types.PasswordResetRequiredException
    var authResult *types.AuthenticationResultType
    signedIn := false
    for !signedIn && resetRequired == nil {
        log.Printf("Signing in to Cognito as user '%v'. The expected result is a
        PasswordResetRequiredException.\n\n", user.UserName)
        authResult, err = runner.cognitoActor.SignIn(clientId, user.UserName, "_")
        if err != nil {
            if errors.As(err, &resetRequired) {
                log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
                DynamoDB known users table.\n"+
                    "User migration is started and a password reset is required.",
                user.UserName)
            } else {
                panic(err)
            }
        } else {
            log.Printf("User '%v' successfully signed in. This is unexpected and probably
            means you have not\n"+
                "cleaned up a previous run of this scenario, so the user exist in the Cognito
            user pool.\n"+
                "You can continue this example and select to clean up resources, or manually
            remove\n"+
                "the user from your user pool and try again.", user.UserName)
            runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
            *authResult.AccessToken)
        }
    }
}
```

```

    signedIn = true
  }
}

log.Println(strings.Repeat("-", 88))
return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(clientId string, user actions.User) {
    wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
to Cognito, you must be able to receive a confirmation\n"+
"code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
    if !wantCode {
        log.Println("To complete this example and successfully migrate a user to
Cognito, you must enter an email\n" +
"you own that can receive a confirmation code.")
        return
    }
    codeDelivery, err := runner.cognitoActor.ForgotPassword(clientId, user.UserName)
    if err != nil {
        panic(err)
    }
    log.Printf("\nA confirmation code has been sent to %v.",
*codeDelivery.Destination)
    code := runner.questioner.Ask("Check your email and enter it here:")

    confirmed := false
    password := runner.questioner.AskPassword("\nEnter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
    for !confirmed {
        log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
        err = runner.cognitoActor.ConfirmForgotPassword(clientId, code, user.UserName,
password)
        if err != nil {
            var invalidPassword *types.InvalidPasswordException
            if errors.As(err, &invalidPassword) {
                password = runner.questioner.AskPassword("\nEnter another password:", 8)
            } else {
                panic(err)
            }
        } else {
            confirmed = true
        }
    }
}

```

```

    }
}
log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
log.Println("Signing in with your username and password...")
authResult, err := runner.cognitoActor.SignIn(clientId, user.UserName, password)
if err != nil {
    panic(err)
}
log.Printf("Successfully signed in. Your access token starts with: %v...\n",
(*authResult.AccessToken)[:10])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)

log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup()
        }
    }()

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome\n")

    log.Println(strings.Repeat("-", 88))

    stackOutputs, err := runner.helper.GetStackOutputs(stackName)
    if err != nil {
        panic(err)
    }
    runner.resources.userPoolId = stackOutputs["UserPoolId"]

    runner.AddMigrateUserTrigger(stackOutputs["UserPoolId"],
stackOutputs["MigrateUserFunctionArn"])
    runner.resources.triggers = append(runner.resources.triggers,
actions.UserMigration)
    resetNeeded, user := runner.SignInUser(stackOutputs["TableName"],
stackOutputs["UserPoolClientId"])
    if resetNeeded {
        runner.helper.ListRecentLogEvents(stackOutputs["MigrateUserFunction"])
    }
}

```

```
runner.ResetPassword(stackOutputs["UserPoolClientId"], user)
}

runner.resources.Cleanup()

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}
```

Controle el desencadenador `MigrateUser` con una función de Lambda.

```
const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsMigrateUser)
(events.CognitoEventUserPoolsMigrateUser, error) {
    log.Printf("Received migrate trigger from %v for user '%v'",
event.TriggerSource, event.UserName)
    if event.TriggerSource != "UserMigration_Authentication" {
        return event, nil
    }
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserName: event.UserName,
    }
}
```

```

log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
if err != nil {
    log.Printf("Error building expression to query for user '%v'.\n",
user.UserName)
    return event, err
}
output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
    TableName:          aws.String(tableName),
    FilterExpression:   expr.Filter(),
    ExpressionAttributeNames: expr.Names(),
    ExpressionAttributeValues: expr.Values(),
})
if err != nil {
    log.Printf("Error looking up user '%v'.\n", user.UserName)
    return event, err
}
if output.Items == nil || len(output.Items) == 0 {
    log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
    return event, err
}

var users []UserInfo
err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
if err != nil {
    log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
    return event, err
}

user = users[0]
log.Printf("UserName '%v' found with email %v. User is migrated and must reset
password.\n", user.UserName, user.UserEmail)
event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
map[string]string{
    "email":          user.UserEmail,
    "email_verified": "true", // email_verified is required for the forgot password
flow.
}
event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
"RESET_REQUIRED"
event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

return event, err

```

```

}

func main() {
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
}

```

Cree una estructura que lleve a cabo las tareas habituales.

```

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(stackName string) (actions.StackOutputs, error)
    PopulateUserTable(tableName string)
    GetKnownUsers(tableName string) (actions.UserList, error)
    AddKnownUser(tableName string, user actions.User)
    ListRecentLogEvents(functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor *actions.CloudFormationActions
    cwActor *actions.CloudWatchLogsActions
    isTestRun bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
ScenarioHelper {
    scenario := ScenarioHelper{

```

```
    questioner: questioner,
    dynamoActor: &actions.DynamoActions{DynamoClient:
dynamodb.NewFromConfig(sdkConfig)},
    cfnActor: &actions.CloudFormationActions{CfnClient:
cloudformation.NewFromConfig(sdkConfig)},
    cwlActor: &actions.CloudWatchLogsActions{CwlClient:
cloudwatchlogs.NewFromConfig(sdkConfig)},
}
return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
structured format.
func (helper ScenarioHelper) GetStackOutputs(stackName string)
(actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(tableName string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(tableName)
    if err != nil {
        panic(err)
    }
}

// GetKnownUsers gets the users from the known users table in a structured
format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
error) {
    knownUsers, err := helper.dynamoActor.Scan(tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
tableName, err)
    }
}
```

```
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(tableName string, user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
*logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(functionName,
*logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
    for _, event := range events {
        log.Printf("\t%v", *event.Message)
    }
    log.Println(strings.Repeat("-", 88))
}
```

Cree una estructura que ajuste las acciones de Amazon Cognito.



```
type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(userPoolId string,
    triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
        &cognitoidentityprovider.DescribeUserPoolInput{
            UserPoolId: aws.String(userPoolId),
        })
    if err != nil {
        log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
            userPoolId, err)
        return err
    }
    lambdaConfig := output.UserPool.LambdaConfig
    for _, trigger := range triggers {
        switch trigger.Trigger {
        case PreSignUp:
            lambdaConfig.PreSignUp = trigger.HandlerArn
        case UserMigration:
            lambdaConfig.UserMigration = trigger.HandlerArn
        case PostAuthentication:
```

```
    lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
}
_, err = actor.CognitoClient.UpdateUserPool(context.TODO(),
&cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:  aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
})
if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(clientId string, userName string, password
string, userEmail string) (bool, error) {
  confirmed := false
  output, err := actor.CognitoClient.SignUp(context.TODO(),
&cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
    {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
})
if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
    log.Println(*invalidPassword.Message)
  } else {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
} else {
  confirmed = output.UserConfirmed
}
return confirmed, err
}
```

```
// SignIn signs in a user to Amazon Cognito using a username and password
authentication flow.
func (actor CognitoActions) SignIn(clientId string, userName string, password
string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
&cognitoidentityprovider.InitiateAuthInput{
    AuthFlow:      "USER_PASSWORD_AUTH",
    ClientId:      aws.String(clientId),
    AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
    })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    } else {
        authResult = output.AuthenticationResult
    }
    return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(clientId string, userName string)
(*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
&cognitoidentityprovider.ForgotPasswordInput{
    ClientId: aws.String(clientId),
    Username: aws.String(userName),
    })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
userName, err)
    }
    return output.CodeDeliveryDetails, err
}
```

```
// ConfirmForgotPassword confirms a user with a confirmation code and a new
password.
func (actor CognitoActions) ConfirmForgotPassword(clientId string, code string,
  userName string, password string) error {
  _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
    &cognitoidentityprovider.ConfirmForgotPasswordInput{
      ClientId:      aws.String(clientId),
      ConfirmationCode: aws.String(code),
      Password:      aws.String(password),
      Username:      aws.String(userName),
    })
  if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
      log.Println(*invalidPassword.Message)
    } else {
      log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
    }
  }
  return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(userAccessToken string) error {
  _, err := actor.CognitoClient.DeleteUser(context.TODO(),
    &cognitoidentityprovider.DeleteUserInput{
      AccessToken: aws.String(userAccessToken),
    })
  if err != nil {
    log.Printf("Couldn't delete user. Here's why: %v\n", err)
  }
  return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(userPoolId string, userName string,
  userEmail string) error {
```

```
_, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
&cognitoidentityprovider.AdminCreateUserInput{
    UserPoolId:    aws.String(userPoolId),
    Username:      aws.String(userName),
    MessageAction: types.MessageActionTypeSuppress,
    UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}}},
)})
if err != nil {
    var userExists *types.UsernameExistsException
    if errors.As(err, &userExists) {
        log.Printf("User %v already exists in the user pool.", userName)
        err = nil
    } else {
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
    }
}
return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(userPoolId string, userName
string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
&cognitoidentityprovider.AdminSetUserPasswordInput{
        Password:    aws.String(password),
        UserPoolId:  aws.String(userPoolId),
        Username:    aws.String(userName),
        Permanent:   true,
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
        }
    }
    return err
}
```

```
}
```

Cree una estructura que ajuste las acciones de DynamoDB.

```
// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
    UserName string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId string
    Time string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
// strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}
```

```
// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(tableName string) error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), userEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
        writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}})
    }
    _, err = actor.DynamoClient.BatchWriteItem(context.TODO(),
&dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
    if err != nil {
        log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
tableName, err)
    }
    return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(tableName string) (UserList, error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(context.TODO(), &dynamodb.ScanInput{
    TableName: aws.String(tableName),
})
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}
```

```
// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(tableName string, user User) error {
    userItem, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
    }
    _, err = actor.DynamoClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
        Item:      userItem,
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
    }
    return err
}
```

Cree una estructura que ajuste las acciones de los registros de CloudWatch.

```
type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
(types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
&cloudwatchlogs.DescribeLogStreamsInput{
        Descending:  aws.Bool(true),
        Limit:       aws.Int32(1),
        LogGroupName: aws.String(logGroupName),
        OrderBy:    types.OrderByLastEventTime,
    })
    if err != nil {
        log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
logGroupName, err)
    } else {
        logStream = output.LogStreams[0]
    }
}
```



```

    }
    return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(functionName string,
logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(context.TODO(),
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
logStreamName, err)
} else {
    events = output.Events
}
return events, err
}

```

Cree una estructura que ajuste las acciones de AWS CloudFormation.

```

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(stackName string) StackOutputs {
output, err := actor.CfnClient.DescribeStacks(context.TODO(),
&cloudformation.DescribeStacksInput{

```

```

    StackName: aws.String(stackName),
  })
  if err != nil || len(output.Stacks) == 0 {
    log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
      stackName, err)
  }
  stackOutputs := StackOutputs{}
  for _, out := range output.Stacks[0].Outputs {
    stackOutputs[*out.OutputKey] = *out.OutputValue
  }
  return stackOutputs
}

```

## Eliminación de recursos.

```

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
  userPoolId      string
  userAccessTokens []string
  triggers        []actions.Trigger

  cognitoActor *actions.CognitoActions
  questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
  demotools.IQuestioner) {
  resources.userAccessTokens = []string{}
  resources.triggers = []actions.Trigger{}
  resources.cognitoActor = cognitoActor
  resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup() {
  defer func() {
    if r := recover(); r != nil {
      log.Printf("Something went wrong during cleanup.\n%v\n", r)
    }
  }()
}

```

```
    log.Println("Use the AWS Management Console to remove any remaining resources\n" +\n        "that were created for this scenario.")\n    }\n}\n\nwantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS\nresources that were created "+\n    "during this demo (y/n)?", "y")\nif wantDelete {\n    for _, accessToken := range resources.userAccessTokens {\n        err := resources.cognitoActor.DeleteUser(accessToken)\n        if err != nil {\n            log.Println("Couldn't delete user during cleanup.")\n            panic(err)\n        }\n        log.Println("Deleted user.")\n    }\n    triggerList := make([]actions.TriggerInfo, len(resources.triggers))\n    for i := 0; i < len(resources.triggers); i++ {\n        triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],\nHandlerArn: nil}\n    }\n    err := resources.cognitoActor.UpdateTriggers(resources.userPoolId,\ntriggerList...)\n    if err != nil {\n        log.Println("Couldn't update Cognito triggers during cleanup.")\n        panic(err)\n    }\n    log.Println("Removed Cognito triggers from user pool.")\n} else {\n    log.Println("Be sure to remove resources when you're done with them to avoid\nunexpected charges!")\n}\n}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Go.
  - [ConfirmForgotPassword](#)
  - [DeleteUser](#)

- [ForgotPassword](#)
- [InitiateAuth](#)
- [SignUp](#)
- [UpdateUserPool](#)

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Primeros pasos para la creación y la invocación de funciones de Lambda con un AWS SDK

En el siguiente ejemplo de código, se muestra cómo:

- Cree un rol de IAM y una función de Lambda y, a continuación, cargue el código de controlador.
- Invocar la función con un único parámetro y obtener resultados.
- Actualizar el código de la función y configurar con una variable de entorno.
- Invocar la función con un nuevo parámetro y obtener resultados. Mostrar el registro de ejecución devuelto.
- Enumere las funciones de su cuenta y, luego, limpie los recursos.

Para obtener información, consulte [Crear una función de Lambda con la consola](#).

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree métodos que realicen acciones de Lambda.

```
namespace LambdaActions;
```

```
using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //             the source code is stored.
        // S3Key     - The name of the file containing the code.
        var functionCode = new FunctionCode
```

```
    {
        S3Bucket = s3Bucket,
        S3Key = s3Key,
    };

    var createFunctionRequest = new CreateFunctionRequest
    {
        FunctionName = functionName,
        Description = "Created by the Lambda .NET API",
        Code = functionCode,
        Handler = handler,
        Runtime = Runtime.Dotnet6,
        Role = role,
    };

    var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
    return reponse.FunctionArn;
}

/// <summary>
/// Delete an AWS Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// delete.</param>
/// <returns>A Boolean value that indicates the success of the action.</
returns>
public async Task<bool> DeleteFunctionAsync(string functionName)
{
    var request = new DeleteFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.DeleteFunctionAsync(request);

    // A return value of NoContent means that the request was processed.
    // In this case, the function was deleted, and the return value
    // is intentionally blank.
    return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
}
```

```
/// <summary>
/// Gets information about a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function for
/// which to retrieve information.</param>
/// <returns>Async Task.</returns>
public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
{
    var functionRequest = new GetFunctionRequest
    {
        FunctionName = functionName,
    };

    var response = await _lambdaService.GetFunctionAsync(functionRequest);
    return response.Configuration;
}

/// <summary>
/// Invoke a Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to
/// invoke.</param>
/// <param name="parameters">The parameter values that will be passed to the
function.</param>
/// <returns>A System Threading Task.</returns>
public async Task<string> InvokeFunctionAsync(
    string functionName,
    string parameters)
{
    var payload = parameters;
    var request = new InvokeRequest
    {
        FunctionName = functionName,
        Payload = payload,
    };

    var response = await _lambdaService.InvokeAsync(request);
    MemoryStream stream = response.Payload;
    string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
    return returnValue;
}
```

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}

/// <summary>
/// Update an existing Lambda function.
/// </summary>
/// <param name="functionName">The name of the Lambda function to update.</
param>
/// <param name="bucketName">The bucket where the zip file containing
/// the Lambda function code is stored.</param>
/// <param name="key">The key name of the source code file.</param>
/// <returns>Async Task.</returns>
public async Task UpdateFunctionCodeAsync(
    string functionName,
    string bucketName,
    string key)
{
    var functionCodeRequest = new UpdateFunctionCodeRequest
    {
        FunctionName = functionName,
        Publish = true,
        S3Bucket = bucketName,
        S3Key = key,
    };
};
```



```
        var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
    }

    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
actions.</param>
    /// <param name="environmentVariables">A dictionary of environment
variables.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}
```

Cree una función que ejecute el escenario.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
                        LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
                        LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonLambda>()
                    .AddAWSService<IAmazonIdentityManagementService>()
                    .AddTransient<LambdaWrapper>()
                    .AddTransient<LambdaRoleWrapper>()
                    .AddTransient<UIWrapper>()
            )
            .Build();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
```

```
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<LambdaBasics>();

var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

string functionName = configuration["FunctionName"]!;
string roleName = configuration["RoleName"]!;
string policyDocument = "{" +
    "  \"Version\": \"2012-10-17\", " +
    "  \"Statement\": [ " +
    "    { " +
    "      \"Effect\": \"Allow\", " +
    "      \"Principal\": { " +
    "        \"Service\": \"lambda.amazonaws.com\" " +
    "      }, " +
    "      \"Action\": \"sts:AssumeRole\" " +
    "    } " +
    "  ] " +
    "}";

var incrementHandler = configuration["IncrementHandler"];
var calculatorHandler = configuration["CalculatorHandler"];
var bucketName = configuration["BucketName"];
var incrementKey = configuration["IncrementKey"];
var calculatorKey = configuration["CalculatorKey"];
var policyArn = configuration["PolicyArn"];

uiWrapper.DisplayLambdaBasicsOverview();

// Create the policy to use with the AWS Lambda functions and then attach
the
// policy to a new role.
var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

Console.WriteLine("Waiting for role to become active.");
```

```
    uiWrapper.WaitABit(15, "Wait until the role is active before trying to
use it.");

    // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
    var success = await
lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn, roleName);
    uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to
the role.");

    // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
// (Amazon S3) bucket.
uiWrapper.DisplayTitle("Create Lambda Function");
Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
    functionName,
    bucketName,
    incrementKey,
    roleArn,
    incrementHandler);

Console.WriteLine("Waiting for the new function to be available.");
Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

// Get the Lambda function.
Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
FunctionConfiguration config;
do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.WriteLine(".");
}
while (config.State != State.Active);

Console.WriteLine($"
The function, {functionName} has been created.");
Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

uiWrapper.PressEnter();

// List the Lambda functions.
uiWrapper.DisplayTitle("Listing all Lambda functions.");
var functions = await lambdaWrapper.ListFunctionsAsync();
```

```
DisplayFunctionList(functions);

uiWrapper.DisplayTitle("Invoke increment function");
Console.WriteLine("Now that it has been created, invoke the Lambda
increment function.");
string? value;
do
{
    Console.Write("Enter a value to increment: ");
    value = Console.ReadLine();
}
while (string.IsNullOrEmpty(value));

string functionParameters = "{" +
    "\"action\": \"increment\", " +
    "\"x\": \"" + value + "\"" +
    "}";
var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
Console.WriteLine($"{value} + 1 = {answer}.");

uiWrapper.DisplayTitle("Update function");
Console.WriteLine("Now update the Lambda function code.");
await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.Write(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

await lambdaWrapper.UpdateFunctionConfigurationAsync(
    functionName,
    calculatorHandler,
    new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

do
{
    config = await lambdaWrapper.GetFunctionAsync(functionName);
    Console.Write(".");
}
while (config.LastUpdateStatus == LastUpdateStatus.InProgress);
```

```
uiWrapper.DisplayTitle("Call updated function");
Console.WriteLine("Now call the updated function...");

bool done = false;

do
{
    string? opSelected;

    Console.WriteLine("Select the operation to perform:");
    Console.WriteLine("\t1. add");
    Console.WriteLine("\t2. subtract");
    Console.WriteLine("\t3. multiply");
    Console.WriteLine("\t4. divide");
    Console.WriteLine("\t0r enter \"q\" to quit.");
    Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the
operation you want to perform: ");
    do
    {
        Console.Write("Your choice? ");
        opSelected = Console.ReadLine();
    }
    while (opSelected == string.Empty);

    var operation = (opSelected) switch
    {
        "1" => "add",
        "2" => "subtract",
        "3" => "multiply",
        "4" => "divide",
        "q" => "quit",
        _ => "add",
    };

    if (operation == "quit")
    {
        done = true;
    }
    else
    {
        // Get two numbers and an action from the user.
        value = string.Empty;
        do
```

```
        {
            Console.WriteLine("Enter the first value: ");
            value = Console.ReadLine();
        }
        while (value == string.Empty);

        string? value2;
        do
        {
            Console.WriteLine("Enter a second value: ");
            value2 = Console.ReadLine();
        }
        while (value2 == string.Empty);

        functionParameters = "{" +
            "\"action\": \"" + operation + "\", " +
            "\"x\": \"" + value + "\", " +
            "\"y\": \"" + value2 + "\" +
            "}";

        answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"The answer when we {operation} the two
numbers is: {answer}.");
    }

    uiWrapper.PressEnter();
} while (!done);

// Delete the function created earlier.

uiWrapper.DisplayTitle("Clean up resources");
// Detach the IAM policy from the IAM role.
Console.WriteLine("First detach the IAM policy from the role.");
success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached
from the role.");

Console.WriteLine("Delete the AWS Lambda function.");
success = await lambdaWrapper.DeleteFunctionAsync(functionName);
if (success)
{
    Console.WriteLine($"The {functionName} function was deleted.");
}
```

```
    }
    else
    {
        Console.WriteLine($"Could not remove the function {functionName}");
    }

    // Now delete the IAM role created for use with the functions
    // created by the application.
    Console.WriteLine("Now we can delete the role that we created.");
    success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
    if (success)
    {
        Console.WriteLine("The role has been successfully removed.");
    }
    else
    {
        Console.WriteLine("Couldn't delete the role.");
    }

    Console.WriteLine("The Lambda Scenario is now complete.");
    uiWrapper.PressEnter();

    // Displays a formatted list of existing functions returned by the
    // LambdaMethods.ListFunctions.
    void DisplayFunctionList(List<FunctionConfiguration> functions)
    {
        functions.ForEach(functionConfig =>
        {
            Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
        });
    }
}

namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;
```



```
public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
{
    _lambdaRoleService = lambdaRoleService;
}

/// <summary>
/// Attach an AWS Identity and Access Management (IAM) role policy to the
/// IAM role to be assumed by the AWS Lambda functions created for the
scenario.
/// </summary>
/// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
policy.</param>
/// <param name="roleName">The name of the IAM role to attach the IAM policy
to.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
roleName)
{
    var response = await _lambdaRoleService.AttachRolePolicyAsync(new
AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Create a new IAM role.
/// </summary>
/// <param name="roleName">The name of the IAM role to create.</param>
/// <param name="policyDocument">The policy document for the new IAM role.</
param>
/// <returns>A string representing the ARN for newly created role.</returns>
public async Task<string> CreateLambdaRoleAsync(string roleName, string
policyDocument)
{
    var request = new CreateRoleRequest
    {
        AssumeRolePolicyDocument = policyDocument,
        RoleName = roleName,
    };

    var response = await _lambdaRoleService.CreateRoleAsync(request);
    return response.Role.Arn;
}
```

```
    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

namespace LambdaScenarioCommon;
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management
(IAM) role that will be assumed by the functions we create.");
    }
}
```

```
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the
value passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a
value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
```

```

    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}

```

Defina un controlador de Lambda que aumente un número.

```

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace LambdaIncrement;

public class Function
{
    /// <summary>
    /// A simple function increments the integer parameter.

```

```

    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}

```

Defina un segundo controlador de Lambda que realice operaciones aritméticas.

```

using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
[assembly:
LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSer

namespace LambdaCalculator;

public class Function
{

    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>

```

```
/// <param name="input">JSON data containing an action, and x and y values.
/// Valid actions include: add, subtract, multiply, and divide.</param>
/// <param name="context">The context object passed by Lambda containing
/// information about invocation, function, and execution environment.</
param>
/// <returns>A string representing the results of the calculation.</returns>
public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
context)
{
    var action = input["action"];
    int x = Convert.ToInt32(input["x"]);
    int y = Convert.ToInt32(input["y"]);
    int result;
    switch (action)
    {
        case "add":
            result = x + y;
            break;
        case "subtract":
            result = x - y;
            break;
        case "multiply":
            result = x * y;
            break;
        case "divide":
            if (y == 0)
            {
                Console.Error.WriteLine("Divide by zero error.");
                result = 0;
            }
            else
                result = x / y;
            break;
        default:
            Console.Error.WriteLine($"{action} is not a valid operation.");
            result = 0;
            break;
    }
    return result;
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for .NET.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## C++

### SDK para C++

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
#!/ Get started with functions scenario.
/*!
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Lambda::getStartedWithFunctionsScenario(
    const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::Lambda::LambdaClient client(clientConfig);

    // 1. Create an AWS Identity and Access Management (IAM) role for Lambda
    function.
    Aws::String roleArn;
    if (!getIamRoleArn(roleArn, clientConfig)) {
        return false;
    }

    // 2. Create a Lambda function.
```

```
int seconds = 0;
do {
    Aws::Lambda::Model::CreateFunctionRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
    request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
    request.SetTimeout(15);
    request.SetMemorySize(128);

    // Assume the AWS Lambda function was built in Docker with same
    architecture
    // as this code.
#if defined(__x86_64__)
    request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
    request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
    request.SetRuntime(Aws::Lambda::Model::Runtime::python3_8);
#endif

    request.SetRole(roleArn);
    request.SetHandler(LAMBDA_HANDLER_NAME);
    request.SetPublish(true);
    Aws::Lambda::Model::FunctionCode code;
    std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                           std::ios_base::in | std::ios_base::binary);
    if (!ifstream.is_open()) {
        std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
        std::endl;
    }

#if USE_CPP_LAMBDA_FUNCTION
    std::cerr
        << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
        << std::endl;
#endif

    deleteIamRole(clientConfig);
    return false;
}

Aws::StringStream buffer;
```



```

    buffer << ifstream.rdbuf();

    code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                                    buffer.str().length()));
    request.SetCode(code);

    Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda function was successfully created. " <<
seconds
                << " seconds elapsed." << std::endl;
        break;
    }
    else if (outcome.GetError().GetErrorType() ==
            Aws::Lambda::LambdaErrors::INVALID_PARAMETER_VALUE &&
            outcome.GetError().GetMessage().find("role") >= 0) {
        if ((seconds % 5) == 0) { // Log status every 10 seconds.
            std::cout
                << "Waiting for the IAM role to become available as a
CreateFunction parameter. "
                << seconds
                << " seconds elapsed." << std::endl;

            std::cout << outcome.GetError().GetMessage() << std::endl;
        }
    }
    else {
        std::cerr << "Error with CreateFunction. "
                << outcome.GetError().GetMessage()
                << std::endl;
        deleteIamRole(clientConfig);
        return false;
    }
    ++seconds;
    std::this_thread::sleep_for(std::chrono::seconds(1));
} while (60 > seconds);

std::cout << "The current Lambda function increments 1 by an input." <<
std::endl;

```

```

// 3. Invoke the Lambda function.
{
    int increment = askQuestionForInt("Enter an increment integer: ");

    Aws::Lambda::Model::InvokeResult invokeResult;
    Aws::Utils::Json::JsonValue jsonPayload;
    jsonPayload.WithString("action", "increment");
    jsonPayload.WithInteger("number", increment);
    if (invokeLambdaFunction(jsonPayload, Aws::Lambda::Model::LogType::Tail,
                            invokeResult, client)) {
        Aws::Utils::Json::JsonValue jsonValue(invokeResult.GetPayload());
        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
            jsonValue.View().GetAllObjects();
        auto iter = values.find("result");
        if (iter != values.end() && iter->second.IsIntegerType()) {
            {
                std::cout << INCREMENT_RESULT_PREFIX
                    << iter->second.AsInteger() << std::endl;
            }
        }
        else {
            std::cout << "There was an error in execution. Here is the log."
                << std::endl;
            Aws::Utils::ByteBuffer buffer =
                Aws::Utils::HashingUtils::Base64Decode(
                    invokeResult.GetLogResult());
            std::cout << "With log " << buffer.GetUnderlyingData() <<
                std::endl;
        }
    }
}

std::cout
    << "The Lambda function will now be updated with new code. Press
return to continue, ";
    Aws::String answer;
    std::getline(std::cin, answer);

// 4. Update the Lambda function code.
{
    Aws::Lambda::Model::UpdateFunctionCodeRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                            std::ios_base::in | std::ios_base::binary);

```

```
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                << "The cpp Lambda function must be built following the
instructions in the cpp_lambda/README.md file. "
                << std::endl;
#endif

            deleteLambdaFunction(client);
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
            Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
buffer.str().length()));

        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
            request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
std::endl;
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                << outcome.GetError().GetMessage()
                << std::endl;
        }
    }

    std::cout
        << "This function uses an environment variable to control the logging
level."
        << std::endl;
    std::cout
        << "UpdateFunctionConfiguration will be used to set the LOG_LEVEL to
DEBUG."
```

```

        << std::endl;
seconds = 0;

// 5. Update the Lambda function configuration.
do {
    ++seconds;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
    request.SetFunctionName(LAMBDA_NAME);
    Aws::Lambda::Model::Environment environment;
    environment.AddVariables("LOG_LEVEL", "DEBUG");
    request.SetEnvironment(environment);

    Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda configuration was successfully updated."
        << std::endl;
        break;
    }

    // RESOURCE_IN_USE: function code update not completed.
    else if (outcome.GetError().GetErrorType() !=
        Aws::Lambda::LambdaErrors::RESOURCE_IN_USE) {
        if ((seconds % 10) == 0) { // Log status every 10 seconds.
            std::cout << "Lambda function update in progress . After " <<
seconds
                << " seconds elapsed." << std::endl;
        }
    }
    else {
        std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
        << outcome.GetError().GetMessage()
        << std::endl;
    }

} while (0 < seconds);

if (0 > seconds) {
    std::cerr << "Function failed to become active." << std::endl;
}
else {

```

```

        std::cout << "Updated function active after " << seconds << " seconds."
                << std::endl;
    }

    std::cout
        << "\nThe new code applies an arithmetic operator to two variables, x
an y."
        << std::endl;
    std::vector<Aws::String> operators = {"plus", "minus", "times", "divided-
by"};
    for (size_t i = 0; i < operators.size(); ++i) {
        std::cout << "    " << i + 1 << " " << operators[i] << std::endl;
    }

    // 6. Invoke the updated Lambda function.
    do {
        int operatorIndex = askQuestionForIntRange("Select an operator index 1 -
4 ", 1,
                                                4);
        int x = askQuestionForInt("Enter an integer for the x value ");
        int y = askQuestionForInt("Enter an integer for the y value ");

        Aws::Utils::Json::JsonValue calculateJsonPayload;
        calculateJsonPayload.WithString("action", operators[operatorIndex - 1]);
        calculateJsonPayload.WithInteger("x", x);
        calculateJsonPayload.WithInteger("y", y);
        Aws::Lambda::Model::InvokeResult calculatedResult;
        if (invokeLambdaFunction(calculateJsonPayload,
                                Aws::Lambda::Model::LogType::Tail,
                                calculatedResult, client)) {
            Aws::Utils::Json::JsonValue jsonValue(calculatedResult.GetPayload());
            Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
                jsonValue.View().GetAllObjects();
            auto iter = values.find("result");
            if (iter != values.end() && iter->second.IsIntegerType()) {
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                    << operators[operatorIndex - 1] << " "
                    << y << " is " << iter->second.AsInteger() <<
std::endl;
            }
            else if (iter != values.end() && iter->second.IsFloatingPointType())
{
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                    << operators[operatorIndex - 1] << " "

```

```

        << y << " is " << iter->second.AsDouble() << std::endl;
    }
    else {
        std::cout << "There was an error in execution. Here is the log."
            << std::endl;
        Aws::Utils::ByteBuffer buffer =
    Aws::Utils::HashingUtils::Base64Decode(
        calculatedResult.GetLogResult());
        std::cout << "With log " << buffer.GetUnderlyingData() <<
    std::endl;
    }
}

    answer = askQuestion("Would you like to try another operation? (y/n) ");
} while (answer == "y");

    std::cout
        << "A list of the lambda functions will be retrieved. Press return to
    continue, ";
    std::getline(std::cin, answer);

// 7. List the Lambda functions.

    std::vector<Aws::String> functions;
    Aws::String marker;

    do {
        Aws::Lambda::Model::ListFunctionsRequest request;
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
            request);

        if (outcome.IsSuccess()) {
            const Aws::Lambda::Model::ListFunctionsResult &result =
    outcome.GetResult();
            std::cout << result.GetFunctions().size()
                << " lambda functions were retrieved." << std::endl;

            for (const Aws::Lambda::Model::FunctionConfiguration
    &functionConfiguration: result.GetFunctions()) {
                functions.push_back(functionConfiguration.GetFunctionName());
            }
        }
    } while (true);
}

```

```

        std::cout << functions.size() << " "
                << functionConfiguration.GetDescription() << std::endl;
        std::cout << " "
                <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                functionConfiguration.GetRuntime()) << ": "
                << functionConfiguration.GetHandler()
                << std::endl;
    }
    marker = result.GetNextMarker();
}
else {
    std::cerr << "Error with Lambda::ListFunctions. "
                << outcome.GetError().GetMessage()
                << std::endl;
}
} while (!marker.empty());

// 8. Get a Lambda function.
if (!functions.empty()) {
    std::stringstream question;
    question << "Choose a function to retrieve between 1 and " <<
functions.size()
                << " ";
    int functionIndex = askQuestionForIntRange(question.str(), 1,
static_cast<int>(functions.size()));

    Aws::String functionName = functions[functionIndex - 1];

    Aws::Lambda::Model::GetFunctionRequest request;
    request.SetFunctionName(functionName);

    Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

    if (outcome.IsSuccess()) {
        std::cout << "Function retrieve.\n" <<
outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
                << std::endl;
    }
    else {
        std::cerr << "Error with Lambda::GetFunction. "

```

```

        << outcome.GetError().GetMessage()
        << std::endl;
    }
}

std::cout << "The resources will be deleted. Press return to continue, ";
std::getline(std::cin, answer);

// 9. Delete the Lambda function.
bool result = deleteLambdaFunction(client);

// 10. Delete the IAM role.
return result && deleteIamRole(clientConfig);
}

//! Routine which invokes a Lambda function and returns the result.
/*!
 \param jsonPayload: Payload for invoke function.
 \param logType: Log type setting for invoke function.
 \param invokeResult: InvokeResult object to receive the result.
 \param client: Lambda client.
 \return bool: Successful completion.
 */
bool
AwsDoc::Lambda::invokeLambdaFunction(const Aws::Utils::Json::JsonValue
&jsonPayload,
                                     Aws::Lambda::Model::LogType logType,
                                     Aws::Lambda::Model::InvokeResult
&invokeResult,
                                     const Aws::Lambda::LambdaClient &client) {
    int seconds = 0;
    bool result = false;
    /*
     * In this example, the Invoke function can be called before recently created
     resources are
     * available. The Invoke function is called repeatedly until the resources
     are
     * available.
     */
    do {
        Aws::Lambda::Model::InvokeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetLogType(logType);
    }
}

```



```

    std::shared_ptr<Aws::IOStream> payload =
    Aws::MakeShared<Aws::StringStream>(
        "FunctionTest");
    *payload << jsonPayload.View().WriteReadable();
    request.SetBody(payload);
    request.SetContentType("application/json");
    Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

    if (outcome.IsSuccess()) {
        invokeResult = std::move(outcome.GetResult());
        result = true;
        break;
    }

    // ACCESS_DENIED: because the role is not available yet.
    // RESOURCE_CONFLICT: because the Lambda function is being created or
updated.
    else if ((outcome.GetError().GetErrorType() ==
        Aws::Lambda::LambdaErrors::ACCESS_DENIED) ||
        (outcome.GetError().GetErrorType() ==
        Aws::Lambda::LambdaErrors::RESOURCE_CONFLICT)) {
        if ((seconds % 5) == 0) { // Log status every 10 seconds.
            std::cout << "Waiting for the invoke api to be available, status
" <<
                ((outcome.GetError().GetErrorType() ==
                Aws::Lambda::LambdaErrors::ACCESS_DENIED ?
                "ACCESS_DENIED" : "RESOURCE_CONFLICT")) << ". " <<
seconds
                << " seconds elapsed." << std::endl;
        }
    }
    else {
        std::cerr << "Error with Lambda::InvokeRequest. "
            << outcome.GetError().GetMessage()
            << std::endl;
        break;
    }
    ++seconds;
    std::this_thread::sleep_for(std::chrono::seconds(1));
} while (seconds < 60);

return result;
}

```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for C++.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

Go

SDK para Go V2

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Cree una situación interactiva que le muestre cómo empezar a usar las funciones de Lambda.

```
// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
// following
// actions:
//
// 1. Create an AWS Identity and Access Management (IAM) role and Lambda
//    function, then upload handler code.
// 2. Invoke the function with a single parameter and get results.
// 3. Update the function code and configure with an environment variable.
// 4. Invoke the function with new parameters and get results. Display the
//    returned execution log.
// 5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
    sdkConfig      aws.Config
    functionWrapper actions.FunctionWrapper
```

```
questioner    demotools.IQuestioner
helper        IScenarioHelper
isTestRun     bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario
// instance from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for
// the actions
// used in the scenario.
func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
demotools.IQuestioner,
helper IScenarioHelper) GetStartedFunctionsScenario {
lambdaClient := lambda.NewFromConfig(sdkConfig)
return GetStartedFunctionsScenario{
sdkConfig:      sdkConfig,
functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
questioner:     questioner,
helper:         helper,
}
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run() {
defer func() {
if r := recover(); r != nil {
log.Printf("Something went wrong with the demo.\n")
}
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the AWS Lambda get started with functions demo.")
log.Println(strings.Repeat("-", 88))

role := scenario.GetOrCreateRole()
funcName := scenario.CreateFunction(role)
scenario.InvokeIncrement(funcName)
scenario.UpdateFunction(funcName)
scenario.InvokeCalculator(funcName)
scenario.ListFunctions()
scenario.Cleanup(role, funcName)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
```

```
log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
// does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
// role
// is returned.
func (scenario GetStartedFunctionsScenario) GetOrCreateRole() *iamtypes.Role {
    var role *iamtypes.Role
    iamClient := iam.NewFromConfig(scenario.sdkConfig)
    log.Println("First, we need an IAM role that Lambda can assume.")
    roleName := scenario.questioner.Ask("Enter a name for the role:",
    demotools.NotEmpty{})
    getOutput, err := iamClient.GetRole(context.TODO(), &iam.GetRoleInput{
        RoleName: aws.String(roleName)})
    if err != nil {
        var noSuch *iamtypes.NoSuchEntityException
        if errors.As(err, &noSuch) {
            log.Printf("Role %v doesn't exist. Creating it...\n", roleName)
        } else {
            log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
            roleName, err)
        }
    } else {
        role = getOutput.Role
        log.Printf("Found role %v.\n", *role.RoleName)
    }
    if role == nil {
        trustPolicy := PolicyDocument{
            Version: "2012-10-17",
            Statement: []PolicyStatement{{
                Effect: "Allow",
                Principal: map[string]string{"Service": "lambda.amazonaws.com"},
                Action: []string{"sts:AssumeRole"},
            }},
        }
        policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        createOutput, err := iamClient.CreateRole(context.TODO(), &iam.CreateRoleInput{
            AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
            RoleName: aws.String(roleName),
        })
        if err != nil {
```

```

    log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
}
role = createOutput.Role
_, err = iamClient.AttachRolePolicy(context.TODO(), &iam.AttachRolePolicyInput{
    PolicyArn: aws.String(policyArn),
    RoleName:  aws.String(roleName),
})
if err != nil {
    log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
err)
}
log.Printf("Created role %v.\n", *role.RoleName)
log.Println("Let's give AWS a few seconds to propagate resources...")
scenario.helper.Pause(10)
}
log.Println(strings.Repeat("-", 88))
return role
}

// CreateFunction creates a Lambda function and uploads a handler written in
// Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(role *iamtypes.Role)
string {
    log.Println("Let's create a function that increments a number.\n" +
        "The function uses the 'lambda_handler_basic.py' script found in the\n" +
        "'handlers' directory of this project.")
    funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
demotools.NotEmpty{})
    zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
fmt.Sprintf("%v.py", funcName))
    log.Printf("Creating function %v and waiting for it to be ready.", funcName)
    funcState := scenario.functionWrapper.CreateFunction(funcName,
fmt.Sprintf("%v.lambda_handler", funcName),
        role.Arn, zipPackage)
    log.Printf("Your function is %v.", funcState)
    log.Println(strings.Repeat("-", 88))
    return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The
// function
// parameters are contained in a Go struct that is used to serialize the
// parameters to

```

```

// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int
value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(funcName string) {
    parameters := actions.IncrementParameters{Action: "increment"}
    log.Println("Let's invoke our function. This function increments a number.")
    parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
demotools.NotEmpty{})
    log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
    invokeOutput := scenario.functionWrapper.Invoke(funcName, parameters, false)
    var payload actions.LambdaResultInt
    err := json.Unmarshal(invokeOutput.Payload, &payload)
    if err != nil {
        log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
            funcName, err)
    }
    log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
payload)
    log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(funcName string) {
    log.Println("Let's update the function to an arithmetic calculator.\n" +
        "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
        "'handlers' directory of this project.")
    scenario.questioner.Ask("Press Enter when you're ready.")
    log.Println("Creating deployment package...")
    zipPackage :=
scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
    fmt.Sprintf("%v.py", funcName))
    log.Println("...and updating the Lambda function and waiting for it to be
ready.")
    funcState := scenario.functionWrapper.UpdateFunctionCode(funcName, zipPackage)
    log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
    log.Println("This function uses an environment variable to control logging
level.")
    log.Println("Let's set it to DEBUG to get the most logging.")
    scenario.functionWrapper.UpdateFunctionConfiguration(funcName,

```

```
map[string]string{"LOG_LEVEL": "DEBUG"})
log.Println(strings.Repeat("-", 88))
}

// InvokeCalculator invokes the Lambda calculator function. The parameters are
// stored in a
// Go struct that is used to serialize the parameters to a JSON payload. That
// payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
// either an
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(funcName string) {
    wantInvoke := true
    choices := []string{"plus", "minus", "times", "divided-by"}
    for wantInvoke {
        choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
            choices)
        x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
        y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
        log.Printf("Invoking %v %v %v...", x, choices[choice], y)
        calcParameters := actions.CalculatorParameters{
            Action: choices[choice],
            X:      x,
            Y:      y,
        }
        invokeOutput := scenario.functionWrapper.Invoke(funcName, calcParameters, true)
        var payload any
        if choice == 3 { // divide-by results in a float.
            payload = actions.LambdaResultFloat{}
        } else {
            payload = actions.LambdaResultInt{}
        }
        err := json.Unmarshal(invokeOutput.Payload, &payload)
        if err != nil {
            log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
                funcName, err)
        }
        log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
            calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
        scenario.questioner.Ask("Press Enter to see the logs from the call.")
        logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
        if err != nil {
            log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
        }
    }
}
```

```
}
log.Println(string(logRes))
wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/n)", "y")
}
log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
func (scenario GetStartedFunctionsScenario) ListFunctions() {
    count := scenario.questioner.AskInt(
        "Let's list functions for your account. How many do you want to see?",
        demotools.NotEmpty{})
    functions := scenario.functionWrapper.ListFunctions(count)
    log.Printf("Found %v functions:", len(functions))
    for _, function := range functions {
        log.Printf("\t%v", *function.FunctionName)
    }
    log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(role *iamtypes.Role, funcName
string) {
    if scenario.questioner.AskBool("Do you want to clean up resources created for
this example? (y/n)",
        "y") {
        iamClient := iam.NewFromConfig(scenario.sdkConfig)
        policiesOutput, err := iamClient.ListAttachedRolePolicies(context.TODO(),
            &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
        if err != nil {
            log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
                *role.RoleName, err)
        }
        for _, policy := range policiesOutput.AttachedPolicies {
            _, err = iamClient.DetachRolePolicy(context.TODO(),
                &iam.DetachRolePolicyInput{
                    PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
                })
            if err != nil {
                log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
                    *policy.PolicyArn, *role.RoleName, err)
            }
        }
    }
}
```



```

_, err = iamClient.DeleteRole(context.TODO(), &iam.DeleteRoleInput{RoleName:
role.RoleName})
if err != nil {
    log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
}
log.Printf("Deleted role %v.\n", *role.RoleName)

scenario.functionWrapper.DeleteFunction(funcName)
log.Printf("Deleted function %v.\n", funcName)
} else {
    log.Println("Okay. Don't forget to delete the resources when you're done with
them.")
}
}

```

Cree una estructura que ajuste las acciones individuales de Lambda.

```

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
    LambdaClient *lambda.Client
}

// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(functionName string) types.State {
    var state types.State
    funcOutput, err := wrapper.LambdaClient.GetFunction(context.TODO(),
&lambda.GetFunctionInput{
    FunctionName: aws.String(functionName),
})
    if err != nil {
        log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
    } else {
        state = funcOutput.Configuration.State
    }
    return state
}

```

```
// CreateFunction creates a new Lambda function from code contained in the
// zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
// until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(functionName string, handlerName
string,
iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
var state types.State
_, err := wrapper.LambdaClient.CreateFunction(context.TODO(),
&lambda.CreateFunctionInput{
Code:          &types.FunctionCode{ZipFile: zipPackage.Bytes()},
FunctionName:  aws.String(functionName),
Role:          iamRoleArn,
Handler:       aws.String(handlerName),
Publish:       true,
Runtime:       types.RuntimePython38,
})
if err != nil {
var resConflict *types.ResourceConflictException
if errors.As(err, &resConflict) {
log.Printf("Function %v already exists.\n", functionName)
state = types.StateActive
} else {
log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
}
} else {
waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
funcOutput, err := waiter.WaitForOutput(context.TODO(),
&lambda.GetFunctionInput{
FunctionName:  aws.String(functionName)}, 1*time.Minute)
if err != nil {
log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
} else {
state = funcOutput.Configuration.State
}
}
return state
}
```

```
}

// UpdateFunctionCode updates the code for the Lambda function specified by
// functionName.
// The existing code for the Lambda function is entirely replaced by the code in
// the
// zipPackage buffer. After the update action is called, a
// lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(functionName string, zipPackage
*bytes.Buffer) types.State {
    var state types.State
    _, err := wrapper.LambdaClient.UpdateFunctionCode(context.TODO(),
&lambda.UpdateFunctionCodeInput{
    FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
})
    if err != nil {
        log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
functionName, err)
    } else {
        waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
        funcOutput, err := waiter.WaitForOutput(context.TODO(),
&lambda.GetFunctionInput{
        FunctionName: aws.String(functionName)}, 1*time.Minute)
        if err != nil {
            log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
        } else {
            state = funcOutput.Configuration.State
        }
    }
    return state
}

// UpdateFunctionConfiguration updates a map of environment variables configured
// for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(functionName string,
envVars map[string]string) {
```

```
_, err := wrapper.LambdaClient.UpdateFunctionConfiguration(context.TODO(),
&lambda.UpdateFunctionConfigurationInput{
    FunctionName: aws.String(functionName),
    Environment: &types.Environment{Variables: envVars},
})
if err != nil {
    log.Panicf("Couldn't update configuration for %v. Here's why: %v",
functionName, err)
}
}

// ListFunctions lists up to maxItems functions for the account. This function
uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(maxItems int)
[]types.FunctionConfiguration {
    var functions []types.FunctionConfiguration
    paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
&lambda.ListFunctionsInput{
        MaxItems: aws.Int32(int32(maxItems)),
    })
    for paginator.HasMorePages() && len(functions) < maxItems {
        pageOutput, err := paginator.NextPage(context.TODO())
        if err != nil {
            log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
        }
        functions = append(functions, pageOutput.Functions...)
    }
    return functions
}

// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(functionName string) {
    _, err := wrapper.LambdaClient.DeleteFunction(context.TODO(),
&lambda.DeleteFunctionInput{
        FunctionName: aws.String(functionName),
    })
    if err != nil {
        log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
    }
}
```

```
}

// Invoke invokes the Lambda function specified by functionName, passing the
// parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
// tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(functionName string, parameters any, getLog
bool) *lambda.InvokeOutput {
    logType := types.LogTypeNone
    if getLog {
        logType = types.LogTypeTail
    }
    payload, err := json.Marshal(parameters)
    if err != nil {
        log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
    }
    invokeOutput, err := wrapper.LambdaClient.Invoke(context.TODO(),
&lambda.InvokeInput{
        FunctionName: aws.String(functionName),
        LogType:      logType,
        Payload:      payload,
    })
    if err != nil {
        log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
    }
    return invokeOutput
}

// IncrementParameters is used to serialize parameters to the increment Lambda
// handler.
type IncrementParameters struct {
    Action string `json:"action"`
    Number int    `json:"number"`
}

// CalculatorParameters is used to serialize parameters to the calculator Lambda
// handler.
type CalculatorParameters struct {
    Action string `json:"action"`
```

```
X    int    `json:"x"`
Y    int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
    Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda
handler.
type LambdaResultFloat struct {
    Result float32 `json:"result"`
}
```

Cree una estructura que implemente funciones para ayudar a ejecutar la situación.

```
// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
    Pause(secs int)
    CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
    HandlerPath string
}

// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
    time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
file. The
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can
be
// used to pass a []byte to Lambda when creating the function.
```

```
// The specified destinationFile is the name to give the file when it's deployed
// to Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
destinationFile string) *bytes.Buffer {
    var err error
    buffer := &bytes.Buffer{}
    writer := zip.NewWriter(buffer)
    zFile, err := writer.Create(destinationFile)
    if err != nil {
        log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
destinationFile, err)
    }
    sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
sourceFile))
    if err != nil {
        log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
sourceFile, err)
    } else {
        _, err = zFile.Write(sourceBody)
        if err != nil {
            log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
sourceFile, err)
        }
    }
    err = writer.Close()
    if err != nil {
        log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
    }
    return buffer
}
```

Defina un controlador de Lambda que aumente un número.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
```

Accepts an action and a single number, performs the specified action on the number,  
and returns the result. The only allowable action is 'increment'.

:param event: The event dict that contains the parameters sent when the function

is invoked.

:param context: The context in which the function is called.

:return: The result of the action.

"""

```

result = None
action = event.get("action")
if action == "increment":
    result = event.get("number", 0) + 1
    logger.info("Calculated result of %s", result)
else:
    logger.error("%s is not a valid action.", action)

response = {"result": result}
return response

```

Defina un segundo controlador de Lambda que realice operaciones aritméticas.

```

import logging
import os

logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}

def lambda_handler(event, context):

```



```
"""
    Accepts an action and two numbers, performs the specified action on the
    numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
"""

# Set the log level based on a variable configured in the Lambda environment.
logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
logger.debug("Event: %s", event)

action = event.get("action")
func = ACTIONS.get(action)
x = event.get("x")
y = event.get("y")
result = None
try:
    if func is not None and x is not None and y is not None:
        result = func(x, y)
        logger.info("%s %s %s is %s", x, action, y, result)
    else:
        logger.error("I can't calculate %s %s %s.", x, action, y)
except ZeroDivisionError:
    logger.warning("I can't divide %s by 0!", x)

response = {"result": result}
return response
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Go.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)

- [ListFunctions](#)
- [UpdateFunctionCode](#)
- [UpdateFunctionConfiguration](#)

## Java

### SDK para Java 2.x

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
/*
 * Lambda function names appear as:
 *
 * arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
 *
 * To find this value, look at the function in the AWS Management Console.
 *
 * Before running this Java code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 *
 * This example performs the following tasks:
 *
 * 1. Creates an AWS Lambda function.
 * 2. Gets a specific AWS Lambda function.
 * 3. Lists all Lambda functions.
 * 4. Invokes a Lambda function.
 * 5. Updates the Lambda function code and invokes it again.
 * 6. Updates a Lambda function's configuration value.
 * 7. Deletes a Lambda function.
 */

public class LambdaScenario {
```

```

public static final String DASHES = new String(new char[80]).replace("\0",
"-");

public static void main(String[] args) throws InterruptedException {
    final String usage = ""

        Usage:
            <functionName> <filePath> <role> <handler> <bucketName> <key>
\s

        Where:
            functionName - The name of the Lambda function.\s
            filePath - The path to the .zip or .jar where the code is
located.\s
            role - The AWS Identity and Access Management (IAM) service
role that has Lambda permissions.\s
            handler - The fully qualified method name (for example,
example.Handler::handleRequest).\s
            bucketName - The Amazon Simple Storage Service (Amazon S3)
bucket name that contains the .zip or .jar used to update the Lambda function's
code.\s
            key - The Amazon S3 key name that represents the .zip or .jar
(for example, LambdaHello-1.0-SNAPSHOT.jar).
        """;

    if (args.length != 6) {
        System.out.println(usage);
        System.exit(1);
    }

    String functionName = args[0];
    String filePath = args[1];
    String role = args[2];
    String handler = args[3];
    String bucketName = args[4];
    String key = args[5];

    Region region = Region.US_WEST_2;
    LambdaClient awsLambda = LambdaClient.builder()
        .region(region)
        .build();

    System.out.println(DASHES);
    System.out.println("Welcome to the AWS Lambda example scenario.");

```

```
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("1. Create an AWS Lambda function.");
String funArn = createLambdaFunction(awsLambda, functionName, filePath,
role, handler);
System.out.println("The AWS Lambda ARN is " + funArn);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("2. Get the " + functionName + " AWS Lambda
function.");
getFunction(awsLambda, functionName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. List all AWS Lambda functions.");
listFunctions(awsLambda);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("4. Invoke the Lambda function.");
System.out.println("*** Sleep for 1 min to get Lambda function ready.");
Thread.sleep(60000);
invokeFunction(awsLambda, functionName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("5. Update the Lambda function code and invoke it
again.");
updateFunctionCode(awsLambda, functionName, bucketName, key);
System.out.println("*** Sleep for 1 min to get Lambda function ready.");
Thread.sleep(60000);
invokeFunction(awsLambda, functionName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("6. Update a Lambda function's configuration value.");
updateFunctionConfiguration(awsLambda, functionName, handler);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("7. Delete the AWS Lambda function.");
LambdaScenario.deleteLambdaFunction(awsLambda, functionName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS Lambda scenario completed successfully");
        System.out.println(DASHES);
        awsLambda.close();
    }

    public static String createLambdaFunction(LambdaClient awsLambda,
        String functionName,
        String filePath,
        String role,
        String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            InputStream is = new FileInputStream(filePath);
            SdkBytes fileToUpload = SdkBytes.fromInputStream(is);

            FunctionCode code = FunctionCode.builder()
                .zipFile(fileToUpload)
                .build();

            CreateFunctionRequest functionRequest =
                CreateFunctionRequest.builder()
                    .functionName(functionName)
                    .description("Created by the Lambda Java API")
                    .code(code)
                    .handler(handler)
                    .runtime(Runtime.JAVA8)
                    .role(role)
                    .build();

            // Create a Lambda function using a waiter
            CreateFunctionResponse functionResponse =
                awsLambda.createFunction(functionRequest);
            GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
                .functionName(functionName)
                .build();

            WaiterResponse<GetFunctionResponse> waiterResponse =
                waiter.waitUntilFunctionExists(getFunctionRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            return functionResponse.functionArn();
        }
    }
}
```

```
    } catch (LambdaException | FileNotFoundException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
    return "";
}

public static void getFunction(LambdaClient awsLambda, String functionName) {
    try {
        GetFunctionRequest functionRequest = GetFunctionRequest.builder()
            .functionName(functionName)
            .build();

        GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
        System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void listFunctions(LambdaClient awsLambda) {
    try {
        ListFunctionsResponse functionResult = awsLambda.listFunctions();
        List<FunctionConfiguration> list = functionResult.functions();
        for (FunctionConfiguration config : list) {
            System.out.println("The function name is " +
config.functionName());
        }

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void invokeFunction(LambdaClient awsLambda, String
functionName) {

    InvokeResponse res;
    try {
```

```
// Need a SdkBytes instance for the payload.
JSONObject jsonObj = new JSONObject();
jsonObj.put("inputValue", "2000");
String json = jsonObj.toString();
SdkBytes payload = SdkBytes.fromUtf8String(json);

InvokeRequest request = InvokeRequest.builder()
    .functionName(functionName)
    .payload(payload)
    .build();

res = awsLambda.invoke(request);
String value = res.payload().asUtf8String();
System.out.println(value);

} catch (LambdaException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}

public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
    try {
        LambdaWaiter waiter = awsLambda.waiter();
        UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
            .functionName(functionName)
            .publish(true)
            .s3Bucket(bucketName)
            .s3Key(key)
            .build();

        UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
        GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .build();

        WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
waiter

            .waitUntilFunctionUpdated(getFunctionConfigRequest);
        waiterResponse.matched().response().ifPresent(System.out::println);
```

```
        System.out.println("The last modified value is " +
response.lastModified());

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
    try {
        UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
            .functionName(functionName)
            .handler(handler)
            .runtime(Runtime.JAVA11)
            .build();

        awsLambda.updateFunctionConfiguration(configurationRequest);

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
    try {
        DeleteFunctionRequest request = DeleteFunctionRequest.builder()
            .functionName(functionName)
            .build();

        awsLambda.deleteFunction(request);
        System.out.println("The " + functionName + " function was deleted");

    } catch (LambdaException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
}
```



- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Java 2.x.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Creación de un rol de AWS Identity and Access Management (IAM) que otorga permiso a Lambda para escribir en los registros.

```
log(`Creating role (${NAME_ROLE_LAMBDA})...`);
const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
```

```
const command = new AttachRolePolicyCommand({
  PolicyArn: policyArn,
  RoleName: roleName,
});

return client.send(command);
};
```

Cree una función de Lambda y cargue el código de controlador.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

invoque la función con un único parámetro y obtenga resultados.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

```
};
```

Actualice el código de la función y configure su entorno Lambda con una variable de entorno.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  return client.send(command);
};
```

Enumere las funciones de su cuenta.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Elimine el rol de IAM y la función de Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for JavaScript.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Kotlin

### SDK para Kotlin

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <functionName> <role> <handler> <bucketName> <updatedBucketName>
            <key>

        Where:
            functionName - The name of the AWS Lambda function.
            role - The AWS Identity and Access Management (IAM) service role that
            has AWS Lambda permissions.
            handler - The fully qualified method name (for example,
            example.Handler::handleRequest).
            bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
            name that contains the ZIP or JAR used for the Lambda function's code.
            updatedBucketName - The Amazon S3 bucket name that contains the .zip
            or .jar used to update the Lambda function's code.
            key - The Amazon S3 key name that represents the .zip or .jar file
            (for example, LambdaHello-1.0-SNAPSHOT.jar).
            """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val functionName = args[0]
    val role = args[1]
    val handler = args[2]
    val bucketName = args[3]
    val updatedBucketName = args[4]
    val key = args[5]
```

```
println("Creating a Lambda function named $functionName.")
val funArn = createScFunction(functionName, bucketName, key, handler, role)
println("The AWS Lambda ARN is $funArn")

// Get a specific Lambda function.
println("Getting the $functionName AWS Lambda function.")
getFunction(functionName)

// List the Lambda functions.
println("Listing all AWS Lambda functions.")
listFunctionsSc()

// Invoke the Lambda function.
println("*** Invoke the Lambda function.")
invokeFunctionSc(functionName)

// Update the AWS Lambda function code.
println("*** Update the Lambda function code.")
updateFunctionCode(functionName, updatedBucketName, key)

// println("*** Invoke the function again after updating the code.")
invokeFunctionSc(functionName)

// Update the AWS Lambda function configuration.
println("Update the run time of the function.")
updateFunctionConfiguration(functionName, handler)

// Delete the AWS Lambda function.
println("Delete the AWS Lambda function.")
delFunction(functionName)
}

suspend fun createScFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }
}
```

```
val request =
    CreateFunctionRequest {
        functionName = myFunctionName
        code = functionCode
        description = "Created by the Lambda Kotlin API"
        handler = myHandler
        role = myRole
        runtime = Runtime.Java8
    }

// Create a Lambda function using a waiter
LambdaClient { region = "us-west-2" }.use { awsLambda ->
    val functionResponse = awsLambda.createFunction(request)
    awsLambda.waitUntilFunctionActive {
        functionName = myFunctionName
    }
    return functionResponse.functionArn.toString()
}

suspend fun getFunction(functionNameVal: String) {
    val functionRequest =
        GetFunctionRequest {
            functionName = functionNameVal
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.getFunction(functionRequest)
        println("The runtime of this Lambda function is
        ${response.configuration?.runtime}")
    }
}

suspend fun listFunctionsSc() {
    val request =
        ListFunctionsRequest {
            maxItems = 10
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.listFunctions(request)
        response.functions?.forEach { function ->
            println("The function name is ${function.functionName}")
        }
    }
}
```

```
    }
  }
}

suspend fun invokeFunctionSc(functionNameVal: String) {
    val json = """"{"inputValue":"1000}""""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            payload = byteArray
            logType = LogType.Tail
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("The function payload is
        ${res.payload?.toString(Charsets.UTF_8)}")
    }
}

suspend fun updateFunctionCode(
    functionNameVal: String?,
    bucketName: String?,
    key: String?,
) {
    val functionCodeRequest =
        UpdateFunctionCodeRequest {
            functionName = functionNameVal
            publish = true
            s3Bucket = bucketName
            s3Key = key
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val response = awsLambda.updateFunctionCode(functionCodeRequest)
        awsLambda.waitUntilFunctionUpdated {
            functionName = functionNameVal
        }
        println("The last modified value is " + response.lastModified)
    }
}

suspend fun updateFunctionConfiguration(
```



```
functionNameVal: String?,
handlerVal: String?,
) {
    val configurationRequest =
        UpdateFunctionConfigurationRequest {
            functionName = functionNameVal
            handler = handlerVal
            runtime = Runtime.Java11
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.updateFunctionConfiguration(configurationRequest)
    }
}


suspend fun delFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Kotlin.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## PHP

## SDK para PHP

 Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
namespace Lambda;

use Aws\S3\S3Client;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithLambda
{
    public function run()
    {
        echo("\n");
        echo("-----\n");
        print("Welcome to the AWS Lambda getting started demo using PHP!\n");
        echo("-----\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $iamService = new IAMService();
        $s3client = new S3Client($clientArgs);
        $lambdaService = new LambdaService();

        echo "First, let's create a role to run our Lambda code.\n";
        $roleName = "test-lambda-role-$uniqid";
        $rolePolicyDocument = "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
                {
```

```

        \ "Effect\ ": \ "Allow\ ",
        \ "Principal\ ": {
            \ "Service\ ": \ "lambda.amazonaws.com\ "
        },
        \ "Action\ ": \ "sts:AssumeRole\ "
    }
]
}";
$role = $iamService->createRole($roleName, $rolePolicyDocument);
echo "Created role {$role['RoleName']}. \n";

$iamService->attachRolePolicy(
    $role['RoleName'],
    "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
);
echo "Attached the AWSLambdaBasicExecutionRole to {$role['RoleName']}.
\n";

echo "\nNow let's create an S3 bucket and upload our Lambda code there.
\n";

$bucketName = "test-example-bucket-$uniqid";
$s3client->createBucket([
    'Bucket' => $bucketName,
]);
echo "Created bucket $bucketName. \n";

$functionName = "doc_example_lambda_$uniqid";
$codeBasic = __DIR__ . "/lambda_handler_basic.zip";
$handler = "lambda_handler_basic";
$file = file_get_contents($codeBasic);
$s3client->putObject([
    'Bucket' => $bucketName,
    'Key' => $functionName,
    'Body' => $file,
]);
echo "Uploaded the Lambda code. \n";

$createLambdaFunction = $lambdaService->createFunction($functionName,
$role, $bucketName, $handler);
// Wait until the function has finished being created.
do {
    $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
} while ($getLambdaFunction['Configuration']['State'] == "Pending");

```

```
    echo "Created Lambda function {$getLambdaFunction['Configuration']
['FunctionName']}.\\n";

    sleep(1);

    echo "\\nOk, let's invoke that Lambda code.\\n";
    $basicParams = [
        'action' => 'increment',
        'number' => 3,
    ];
    /** @var Stream $invokeFunction */
    $invokeFunction = $lambdaService->invoke($functionName, $basicParams)
['Payload'];
    $result = json_decode($invokeFunction->getContents())->result;
    echo "After invoking the Lambda code with the input of
{$basicParams['number']} we received $result.\\n";

    echo "\\nSince that's working, let's update the Lambda code.\\n";
    $codeCalculator = "lambda_handler_calculator.zip";
    $handlerCalculator = "lambda_handler_calculator";
    echo "First, put the new code into the S3 bucket.\\n";
    $file = file_get_contents($codeCalculator);
    $s3client->putObject([
        'Bucket' => $bucketName,
        'Key' => $functionName,
        'Body' => $file,
    ]);
    echo "New code uploaded.\\n";

    $lambdaService->updateFunctionCode($functionName, $bucketName,
    $functionName);
    // Wait for the Lambda code to finish updating.
    do {
        $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
"Successful");
    echo "New Lambda code uploaded.\\n";

    $environment = [
        'Variable' => ['Variables' => ['LOG_LEVEL' => 'DEBUG']],
    ];
    $lambdaService->updateFunctionConfiguration($functionName,
    $handlerCalculator, $environment);
```

```
do {
    $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
    } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !=
"Successful");
    echo "Lambda code updated with new handler and a LOG_LEVEL of DEBUG for
more information.\n";

    echo "Invoke the new code with some new data.\n";
    $calculatorParams = [
        'action' => 'plus',
        'x' => 5,
        'y' => 4,
    ];
    $invokeFunction = $lambdaService->invoke($functionName,
$calculatorParams, "Tail");
    $result = json_decode($invokeFunction['Payload']->getContents())->result;
    echo "Indeed, {$calculatorParams['x']} + {$calculatorParams['y']} does
equal $result.\n";
    echo "Here's the extra debug info: ";
    echo base64_decode($invokeFunction['LogResult']) . "\n";

    echo "\nBut what happens if you try to divide by zero?\n";
    $divZeroParams = [
        'action' => 'divide',
        'x' => 5,
        'y' => 0,
    ];
    $invokeFunction = $lambdaService->invoke($functionName, $divZeroParams,
"Tail");
    $result = json_decode($invokeFunction['Payload']->getContents())->result;
    echo "You get a |$result| result.\n";
    echo "And an error message: ";
    echo base64_decode($invokeFunction['LogResult']) . "\n";

    echo "\nHere's all the Lambda functions you have in this Region:\n";
    $listLambdaFunctions = $lambdaService->listFunctions(5);
    $allLambdaFunctions = $listLambdaFunctions['Functions'];
    $next = $listLambdaFunctions->get('NextMarker');
    while ($next != false) {
        $listLambdaFunctions = $lambdaService->listFunctions(5, $next);
        $next = $listLambdaFunctions->get('NextMarker');
        $allLambdaFunctions = array_merge($allLambdaFunctions,
$listLambdaFunctions['Functions']);
    }
```

```
    }
    foreach ($allLambdaFunctions as $function) {
        echo "{$function['FunctionName']}\n";
    }

    echo "\n\nAnd don't forget to clean up your data!\n";

    $lambdaService->deleteFunction($functionName);
    echo "Deleted Lambda function.\n";
    $iamService->deleteRole($role['RoleName']);
    echo "Deleted Role.\n";
    $deleteObjects = $s3client->listObjectsV2([
        'Bucket' => $bucketName,
    ]);
    $deleteObjects = $s3client->deleteObjects([
        'Bucket' => $bucketName,
        'Delete' => [
            'Objects' => $deleteObjects['Contents'],
        ]
    ]);
    echo "Deleted all objects from the S3 bucket.\n";
    $s3client->deleteBucket(['Bucket' => $bucketName]);
    echo "Deleted the bucket.\n";
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for PHP.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Python

### SDK para Python (Boto3)

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Defina un controlador de Lambda que aumente un número.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
    number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Defina un segundo controlador Lambda que realice operaciones aritméticas.

```
import logging
import os

logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}

def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
    numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
    function
                   is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
```



```
        result = func(x, y)
        logger.info("%s %s %s is %s", x, action, y, result)
    else:
        logger.error("I can't calculate %s %s %s.", x, action, y)
except ZeroDivisionError:
    logger.warning("I can't divide %s by 0!", x)

response = {"result": result}
return response
```

Cree funciones que ajusten las acciones de Lambda.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource

    @staticmethod
    def create_deployment_package(source_file, destination_file):
        """
        Creates a Lambda deployment package in .zip format in an in-memory
        buffer. This
        buffer can be passed directly to Lambda when creating the function.

        :param source_file: The name of the file that contains the Lambda handler
            function.
        :param destination_file: The name to give the file when it's deployed to
        Lambda.
        :return: The deployment package.
        """
        buffer = io.BytesIO()
        with zipfile.ZipFile(buffer, "w") as zipped:
            zipped.write(source_file, destination_file)
        buffer.seek(0)
        return buffer.read()

    def get_iam_role(self, iam_role_name):
        """
        Get an AWS Identity and Access Management (IAM) role.
```

```
:param iam_role_name: The name of the role to retrieve.
:return: The IAM role.
"""
role = None
try:
    temp_role = self.iam_resource.Role(iam_role_name)
    temp_role.load()
    role = temp_role
    logger.info("Got IAM role %s", role.name)
except ClientError as err:
    if err.response["Error"]["Code"] == "NoSuchEntity":
        logger.info("IAM role %s does not exist.", iam_role_name)
    else:
        logger.error(
            "Couldn't get IAM role %s. Here's why: %s: %s",
            iam_role_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
return role

def create_iam_role_for_lambda(self, iam_role_name):
    """
    Creates an IAM role that grants the Lambda function basic permissions. If
a
    role with the specified name already exists, it is used for the demo.

    :param iam_role_name: The name of the role to create.
    :return: The role and a value that indicates whether the role is newly
created.
    """
    role = self.get_iam_role(iam_role_name)
    if role is not None:
        return role, False

    lambda_assume_role_policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {"Service": "lambda.amazonaws.com"},
                "Action": "sts:AssumeRole",
```

```
        }
    ],
}
policy_arn = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

try:
    role = self.iam_resource.create_role(
        RoleName=iam_role_name,
        AssumeRolePolicyDocument=json.dumps(lambda_assume_role_policy),
    )
    logger.info("Created role %s.", role.name)
    role.attach_policy(PolicyArn=policy_arn)
    logger.info("Attached basic execution policy to role %s.", role.name)
except ClientError as error:
    if error.response["Error"]["Code"] == "EntityAlreadyExists":
        role = self.iam_resource.Role(iam_role_name)
        logger.warning("The role %s already exists. Using it.",
iam_role_name)
    else:
        logger.exception(
            "Couldn't create role %s or attach policy %s.",
            iam_role_name,
            policy_arn,
        )
        raise

return role, True

def get_function(self, function_name):
    """
    Gets data about a Lambda function.

    :param function_name: The name of the function.
    :return: The function data.
    """
    response = None
    try:
        response =
self.lambda_client.get_function(FunctionName=function_name)
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            logger.info("Function %s does not exist.", function_name)
        else:
```

```
        logger.error(
            "Couldn't get function %s. Here's why: %s: %s",
            function_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    return response

def create_function(
    self, function_name, handler_name, iam_role, deployment_package
):
    """
    Deploys a Lambda function.

    :param function_name: The name of the Lambda function.
    :param handler_name: The fully qualified name of the handler function.
    This
                        must include the file name and the function name.
    :param iam_role: The IAM role to use for the function.
    :param deployment_package: The deployment package that contains the
    function
                        code in .zip format.
    :return: The Amazon Resource Name (ARN) of the newly created function.
    """
    try:
        response = self.lambda_client.create_function(
            FunctionName=function_name,
            Description="AWS Lambda doc example",
            Runtime="python3.8",
            Role=iam_role.arn,
            Handler=handler_name,
            Code={"ZipFile": deployment_package},
            Publish=True,
        )
        function_arn = response["FunctionArn"]
        waiter = self.lambda_client.get_waiter("function_active_v2")
        waiter.wait(FunctionName=function_name)
        logger.info(
            "Created function '%s' with ARN: '%s'.",
            function_name,
            response["FunctionArn"],
        )
```

```
except ClientError:
    logger.error("Couldn't create function %s.", function_name)
    raise
else:
    return function_arn

def delete_function(self, function_name):
    """
    Deletes a Lambda function.

    :param function_name: The name of the function to delete.
    """
    try:
        self.lambda_client.delete_function(FunctionName=function_name)
    except ClientError:
        logger.exception("Couldn't delete function %s.", function_name)
        raise

def invoke_function(self, function_name, function_params, get_log=False):
    """
    Invokes a Lambda function.

    :param function_name: The name of the function to invoke.
    :param function_params: The parameters of the function as a dict. This
dict
                           is serialized to JSON before it is sent to
Lambda.
    :param get_log: When true, the last 4 KB of the execution log are
included in
                   the response.
    :return: The response from the function invocation.
    """
    try:
        response = self.lambda_client.invoke(
            FunctionName=function_name,
            Payload=json.dumps(function_params),
            LogType="Tail" if get_log else "None",
        )
        logger.info("Invoked function %s.", function_name)
    except ClientError:
        logger.exception("Couldn't invoke function %s.", function_name)
        raise
```

```
        return response

    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
        contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
in
                                .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response

    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
```

```
        logger.error(
            "Couldn't update function configuration %s. Here's why: %s: %s",
            function_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response

def list_functions(self):
    """
    Lists the Lambda functions for the current account.
    """
    try:
        func_paginator = self.lambda_client.get_paginator("list_functions")
        for func_page in func_paginator.paginate():
            for func in func_page["Functions"]:
                print(func["FunctionName"])
                desc = func.get("Description")
                if desc:
                    print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
    except ClientError as err:
        logger.error(
            "Couldn't list functions. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

Cree una función que ejecute el escenario.

```
class UpdateFunctionWaiter(CustomWaiter):
    """A custom waiter that waits until a function is successfully updated."""

    def __init__(self, client):
        super().__init__(
```

```
        "UpdateSuccess",
        "GetFunction",
        "Configuration.LastUpdateStatus",
        {"Successful": WaitState.SUCCESS, "Failed": WaitState.FAILURE},
        client,
    )

    def wait(self, function_name):
        self._wait(FunctionName=function_name)

def run_scenario(lambda_client, iam_resource, basic_file, calculator_file,
lambda_name):
    """
    Runs the scenario.

    :param lambda_client: A Boto3 Lambda client.
    :param iam_resource: A Boto3 IAM resource.
    :param basic_file: The name of the file that contains the basic Lambda
handler.
    :param calculator_file: The name of the file that contains the calculator
Lambda handler.
    :param lambda_name: The name to give resources created for the scenario, such
as the
                        IAM role and the Lambda function.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the AWS Lambda getting started with functions demo.")
    print("-" * 88)

    wrapper = LambdaWrapper(lambda_client, iam_resource)

    print("Checking for IAM role for Lambda...")
    iam_role, should_wait = wrapper.create_iam_role_for_lambda(lambda_name)
    if should_wait:
        logger.info("Giving AWS time to create resources...")
        wait(10)

    print(f"Looking for function {lambda_name}...")
    function = wrapper.get_function(lambda_name)
    if function is None:
        print("Zipping the Python script into a deployment package...")
```



```
    deployment_package = wrapper.create_deployment_package(
        basic_file, f"{lambda_name}.py"
    )
    print(f"...and creating the {lambda_name} Lambda function.")
    wrapper.create_function(
        lambda_name, f"{lambda_name}.lambda_handler", iam_role,
deployment_package
    )
else:
    print(f"Function {lambda_name} already exists.")
print("-" * 88)

print(f"Let's invoke {lambda_name}. This function increments a number.")
action_params = {
    "action": "increment",
    "number": q.ask("Give me a number to increment: ", q.is_int),
}
print(f"Invoking {lambda_name}...")
response = wrapper.invoke_function(lambda_name, action_params)
print(
    f"Incrementing {action_params['number']} resulted in "
    f"{json.load(response['Payload'])}"
)
print("-" * 88)

print(f"Let's update the function to an arithmetic calculator.")
q.ask("Press Enter when you're ready.")
print("Creating a new deployment package...")
deployment_package = wrapper.create_deployment_package(
    calculator_file, f"{lambda_name}.py"
)
print(f"...and updating the {lambda_name} Lambda function.")
update_waiter = UpdateFunctionWaiter(lambda_client)
wrapper.update_function_code(lambda_name, deployment_package)
update_waiter.wait(lambda_name)
print(f"This function uses an environment variable to control logging
level.")
print(f"Let's set it to DEBUG to get the most logging.")
wrapper.update_function_configuration(
    lambda_name, {"LOG_LEVEL": logging.getLevelName(logging.DEBUG)}
)

actions = ["plus", "minus", "times", "divided-by"]
want_invoke = True
```

```

while want_invoke:
    print(f"Let's invoke {lambda_name}. You can invoke these actions:")
    for index, action in enumerate(actions):
        print(f"{index + 1}: {action}")
    action_params = {}
    action_index = q.ask(
        "Enter the number of the action you want to take: ",
        q.is_int,
        q.in_range(1, len(actions)),
    )
    action_params["action"] = actions[action_index - 1]
    print(f"You've chosen to invoke 'x {action_params['action']} y'.")
    action_params["x"] = q.ask("Enter a value for x: ", q.is_int)
    action_params["y"] = q.ask("Enter a value for y: ", q.is_int)
    print(f"Invoking {lambda_name}...")
    response = wrapper.invoke_function(lambda_name, action_params, True)
    print(
        f"Calculating {action_params['x']} {action_params['action']}
{action_params['y']} "
        f"resulted in {json.load(response['Payload'])}"
    )
    q.ask("Press Enter to see the logs from the call.")
    print(base64.b64decode(response["LogResult"]).decode())
    want_invoke = q.ask("That was fun. Shall we do it again? (y/n) ",
q.is_yesno)
    print("-" * 88)

    if q.ask(
        "Do you want to list all of the functions in your account? (y/n) ",
q.is_yesno
    ):
        wrapper.list_functions()
        print("-" * 88)

    if q.ask("Ready to delete the function and role? (y/n) ", q.is_yesno):
        for policy in iam_role.attached_policies.all():
            policy.detach_role(RoleName=iam_role.name)
        iam_role.delete()
        print(f"Deleted role {lambda_name}.")
        wrapper.delete_function(lambda_name)
        print(f"Deleted function {lambda_name}.")

print("\nThanks for watching!")
print("-" * 88)

```

```
if __name__ == "__main__":
    try:
        run_scenario(
            boto3.client("lambda"),
            boto3.resource("iam"),
            "lambda_handler_basic.py",
            "lambda_handler_calculator.py",
            "doc_example_lambda_calculator",
        )
    except Exception:
        logging.exception("Something went wrong with the demo!")
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Ruby

### SDK para Ruby

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Configurar los permisos de IAM de requisitos previos para que una función de Lambda pueda escribir registros.

```
# Get an AWS Identity and Access Management (IAM) role.
#
# @param iam_role_name: The name of the role to retrieve.
# @param action: Whether to create or destroy the IAM apparatus.
# @return: The IAM role.
def manage_iam(iam_role_name, action)
  role_policy = {
    'Version': "2012-10-17",
    'Statement': [
      {
        'Effect': "Allow",
        'Principal': {
          'Service': "lambda.amazonaws.com"
        },
        'Action': "sts:AssumeRole"
      }
    ]
  }
  case action
  when "create"
    role = $iam_client.create_role(
      role_name: iam_role_name,
      assume_role_policy_document: role_policy.to_json
    )
    $iam_client.attach_role_policy(
      {
        policy_arn: "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole",
        role_name: iam_role_name
      }
    )
    $iam_client.wait_until(:role_exists, { role_name: iam_role_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
    @logger.debug("Successfully created IAM role: #{role['role']['arn']}")
    @logger.debug("Enforcing a 10-second sleep to allow IAM role to activate
fully.")
    sleep(10)
    return role, role_policy.to_json
  when "destroy"
    $iam_client.detach_role_policy(
      {
```

```

        policy_arn: "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole",
        role_name: iam_role_name
    }
)
$iam_client.delete_role(
    role_name: iam_role_name
)
@logger.debug("Detached policy & deleted IAM role: #{iam_role_name}")
else
    raise "Incorrect action provided. Must provide 'create' or 'destroy'"
end
rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error creating role or attaching policy:\n
#{e.message}")
end

```

Definir un controlador de Lambda que aumente un número dado como parámetro de invocación.

```

require "logger"

# A function that increments a whole number by one (1) and logs the result.
# Requires a manually-provided runtime parameter, 'number', which must be Int
#
# @param event [Hash] Parameters sent when the function is invoked
# @param context [Hash] Methods and properties that provide information
# about the invocation, function, and execution environment.
# @return incremented_number [String] The incremented number.
def lambda_handler(event:, context:)
    logger = Logger.new($stdout)
    log_level = ENV["LOG_LEVEL"]
    logger.level = case log_level
                   when "debug"
                     Logger::DEBUG
                   when "info"
                     Logger::INFO
                   else
                     Logger::ERROR
                   end

    logger.debug("This is a debug log message.")
    logger.info("This is an info log message. Code executed successfully!")
end

```

```

number = event["number"].to_i
incremented_number = number + 1
logger.info("You provided #{number.round} and it was incremented to
#{incremented_number.round}")
incremented_number.round.to_s
end

```

## Comprimir su función de Lambda en un paquete de implementación.

```

# Creates a Lambda deployment package in .zip format.
# This zip can be passed directly as a string to Lambda when creating the
function.
#
# @param source_file: The name of the object, without suffix, for the Lambda
file and zip.
# @return: The deployment package.
def create_deployment_package(source_file)
  Dir.chdir(File.dirname(__FILE__))
  if File.exist?("lambda_function.zip")
    File.delete("lambda_function.zip")
    @logger.debug("Deleting old zip: lambda_function.zip")
  end
  Zip::File.open("lambda_function.zip", create: true) {
    |zipfile|
    zipfile.add("lambda_function.rb", "#{source_file}.rb")
  }
  @logger.debug("Zipping #{source_file}.rb into: lambda_function.zip.")
  File.read("lambda_function.zip").to_s
rescue StandardError => e
  @logger.error("There was an error creating deployment package:\n
#{e.message}")
end

```

## Crear una nueva función de Lambda.

```

# Deploys a Lambda function.
#
# @param function_name: The name of the Lambda function.
# @param handler_name: The fully qualified name of the handler function. This
#                       must include the file name and the function name.
# @param role_arn: The IAM role to use for the function.

```

```

# @param deployment_package: The deployment package that contains the function
#                               code in .zip format.
# @return: The Amazon Resource Name (ARN) of the newly created function.
def create_function(function_name, handler_name, role_arn, deployment_package)
  response = @lambda_client.create_function({
    role: role_arn.to_s,
    function_name: function_name,
    handler: handler_name,
    runtime: "ruby2.7",
    code: {
      zip_file: deployment_package
    },
    environment: {
      variables: {
        "LOG_LEVEL" => "info"
      }
    }
  })

  @lambda_client.wait_until(:function_active_v2, { function_name:
function_name}) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  response
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error creating #{function_name}:\n #{e.message}")
rescue Aws::Waiters::Errors::WaiterFailed => e
  @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
end

```

Invocar su función de Lambda con parámetros de tiempo de ejecución opcionales.

```

# Invokes a Lambda function.
# @param function_name [String] The name of the function to invoke.
# @param payload [nil] Payload containing runtime parameters.
# @return [Object] The response from the function invocation.
def invoke_function(function_name, payload = nil)
  params = { function_name: function_name }
  params[:payload] = payload unless payload.nil?
  @lambda_client.invoke(params)
rescue Aws::Lambda::Errors::ServiceException => e

```

```
@logger.error("There was an error executing #{function_name}:\n
#{e.message}")
end
```

Actualizar la configuración de su función de Lambda para introducir una nueva variable de entorno.

```
# Updates the environment variables for a Lambda function.
# @param function_name: The name of the function to update.
# @param log_level: The log level of the function.
# @return: Data about the update, including the status.
def update_function_configuration(function_name, log_level)
  @lambda_client.update_function_configuration({
    function_name: function_name,
    environment: {
      variables: {
        "LOG_LEVEL" => log_level
      }
    }
  })

  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name}) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
  end
end
```

Actualizar el código de su función de Lambda con un paquete de implementación diferente que contenga un código diferente.

```
# Updates the code for a Lambda function by submitting a .zip archive that
contains
# the code for the function.

# @param function_name: The name of the function to update.
```



```

# @param deployment_package: The function code to update, packaged as bytes in
#                               .zip format.
# @return: Data about the update, including the status.
def update_function_code(function_name, deployment_package)
  @lambda_client.update_function_code(
    function_name: function_name,
    zip_file: deployment_package
  )
  @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name}) do |w|
    w.max_attempts = 5
    w.delay = 5
  end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
end

```

Enumerar todas las funciones de Lambda existentes mediante el paginador integrado.

```

# Lists the Lambda functions for the current account.
def list_functions
  functions = []
  @lambda_client.list_functions.each do |response|
    response["functions"].each do |function|
      functions.append(function["function_name"])
    end
  end
  functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
  end
end

```

Eliminar una función de Lambda específica.

```

# Deletes a Lambda function.

```

```
# @param function_name: The name of the function to delete.
def delete_function(function_name)
  print "Deleting function: #{function_name}..."
  @lambda_client.delete_function(
    function_name: function_name
  )
  print "Done!".green
rescue Aws::Lambda::Errors::ServiceException => e
  @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
end
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Ruby.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

El Cargo.toml con las dependencias utilizadas en este escenario.

```
[package]
name = "lambda-code-examples"
version = "0.1.0"
edition = "2021"
```

```
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-ec2 = { version = "1.3.0" }
aws-sdk-iam = { version = "1.3.0" }
aws-sdk-lambda = { version = "1.3.0" }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-types = { version = "1.0.1" }
clap = { version = "~4.4", features = ["derive"] }
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
tracing = "0.1.37"
serde_json = "1.0.94"
anyhow = "1.0.71"
uuid = { version = "1.3.3", features = ["v4"] }
lambda_runtime = "0.8.0"
serde = "1.0.164"
```

Un conjunto de utilidades que simplifican las llamadas a Lambda para este escenario. Este archivo es `src/ations.rs` en la caja.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
    delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{"
```

```

    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};

/* Operation describes */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl ToString for Operation {
    fn to_string(&self) -> String {
        match self {
            Operation::Plus => "plus".to_string(),
            Operation::Minus => "minus".to_string(),
            Operation::Times => "times".to_string(),
            Operation::DividedBy => "divided-by".to_string(),
        }
    }
}

```

```

    }
  }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {
            InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
            InvokeArgs::Arithmetic(o, i, j) => {
                let mut map: S::SerializeMap =
serializer.serialize_map(Some(3))?;
                map.serialize_key(&"op".to_string())?;
                map.serialize_value(&o.to_string())?;
                map.serialize_key(&"i".to_string())?;
                map.serialize_value(&i)?;
                map.serialize_key(&"j".to_string())?;
                map.serialize_value(&j)?;
                map.end()
            }
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
    behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#"{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}";

```

```
    }
  ]
}";

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 * scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
pub struct LambdaManager {
  iam_client: aws_sdk_iam::Client,
  lambda_client: aws_sdk_lambda::Client,
  s3_client: aws_sdk_s3::Client,
  lambda_name: String,
  role_name: String,
  bucket: String,
  own_bucket: bool,
}

// These unit type structs provide nominal typing on top of String parameters for
// LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
  pub fn new(
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: LambdaName,
    role_name: RoleName,
    bucket: Bucket,
    own_bucket: OwnBucket,
  ) -> Self {
    Self {
      iam_client,
      lambda_client,
      s3_client,
      lambda_name: lambda_name.0,
      role_name: role_name.0,
      bucket: bucket.0,
      own_bucket: own_bucket.0,
    }
  }
}
```

```

    }
}

/**
 * Load the AWS configuration from the environment.
 * Look up lambda_name and bucket if none are given, or generate a random
name if not present in the environment.
 * If the bucket name is provided, the caller needs to have created the
bucket.
 * If the bucket name is generated, it will be created.
 */
pub async fn load_from_env(lambda_name: Option<String>, bucket:
Option<String>) -> Self {
    let sdk_config = aws_config::load_from_env().await;
    let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
        std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
    }));
    let role_name = RoleName(format!("{}",_role", lambda_name.0));
    let (bucket, own_bucket) =
        match bucket {
            Some(bucket) => (Bucket(bucket), false),
            None => (
                Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                    format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                })),
                true,
            ),
        };

    let s3_client = aws_sdk_s3::Client::new(&sdk_config);

    if own_bucket {
        info!("Creating bucket for demo: {}", bucket.0);
        s3_client
            .create_bucket()
            .bucket(bucket.0.clone())
            .create_bucket_configuration(
                CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                    sdk_config.region().unwrap().as_ref(),
                ))
            .build(),
    }
}

```

```
        )
        .send()
        .await
        .unwrap();
    }

    Self::new(
        aws_sdk_iam::Client::new(&sdk_config),
        aws_sdk_lambda::Client::new(&sdk_config),
        s3_client,
        lambda_name,
        role_name,
        bucket,
        OwnBucket(own_bucket),
    )
}

// snippet-start:[lambda.rust.scenario.prepare_function]
/**
 * Upload function code from a path to a zip file.
 * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
 * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
 */
async fn prepare_function(
    &self,
    zip_file: PathBuf,
    key: Option<String>,
) -> Result<FunctionCode, anyhow::Error> {
    let body = ByteStream::from_path(zip_file).await?;

    let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

    info!("Uploading function code to s3://{}/{}", self.bucket, key);
    let _ = self
        .s3_client
        .put_object()
        .bucket(self.bucket.clone())
        .key(key.clone())
        .body(body)
        .send()
        .await?;

    Ok(FunctionCode::builder())
}
```



```

        .s3_bucket(self.bucket.clone())
        .s3_key(key)
        .build()
    }
    // snippet-end:[lambda.rust.scenario.prepare_function]

    // snippet-start:[lambda.rust.scenario.create_function]
    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::ProvidedAl2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }
}

```

```
// snippet-end:[lambda.rust.scenario.create_function]

/**
 * Create an IAM execution role for the managed Lambda function.
 * If the role already exists, use that instead.
 */
async fn create_role(&self) -> Result<aws_sdk_iam::types::Role,
CreateRoleError> {
    info!("Creating execution role for function");
    let get_role = self
        .iam_client
        .get_role()
        .role_name(self.role_name.clone())
        .send()
        .await;
    if let Ok(get_role) = get_role {
        if let Some(role) = get_role.role {
            return Ok(role);
        }
    }

    let create_role = self
        .iam_client
        .create_role()
        .role_name(self.role_name.clone())
        .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
        .send()
        .await;

    match create_role {
        Ok(create_role) => match create_role.role {
            Some(role) => Ok(role),
            None => Err(CreateRoleError::generic(
                ErrorMetadata::builder()
                    .message("CreateRole returned empty success")
                    .build(),
            )),
        },
        Err(err) => Err(err.into_service_error()),
    }
}

/**
```

```

    * Poll `is_function_ready` with a 1-second delay. It returns when the
    function is ready or when there's an error checking the function's state.
    */
    pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
        info!("Waiting for function");
        while !self.is_function_ready(None).await? {
            info!("Function is not ready, sleeping 1s");
            tokio::time::sleep(Duration::from_secs(1)).await;
        }
        Ok(())
    }

    /**
    * Check if a Lambda function is ready to be invoked.
    * A Lambda function is ready for this scenario when its state is active and
    its LastUpdateStatus is Successful.
    * Additionally, if a sha256 is provided, the function must have that as its
    current code hash.
    * Any missing properties or failed requests will be reported as an Err.
    */
    async fn is_function_ready(
        &self,
        expected_code_sha256: Option<&str>,
    ) -> Result<bool, anyhow::Error> {
        match self.get_function().await {
            Ok(func) => {
                if let Some(config) = func.configuration() {
                    if let Some(state) = config.state() {
                        info!(?state, "Checking if function is active");
                        if !matches!(state, State::Active) {
                            return Ok(false);
                        }
                    }
                }
                match config.last_update_status() {
                    Some(last_update_status) => {
                        info!(?last_update_status, "Checking if function is
ready");

                        match last_update_status {
                            LastUpdateStatus::Successful => {
                                // continue
                            }
                            LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                                return Ok(false);

```

```

        }
        unknown => {
            warn!(
                status_variant = unknown.as_str(),
                "LastUpdateStatus unknown"
            );
            return Err(anyhow!(
                "Unknown LastUpdateStatus, fn config is
{config:?}"
            ));
        }
    }
}
None => {
    warn!("Missing last update status");
    return Ok(false);
}
};
if expected_code_sha256.is_none() {
    return Ok(true);
}
if let Some(code_sha256) = config.code_sha256() {
    return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
}
}
Err(e) => {
    warn!(?e, "Could not get function while waiting");
}
}
Ok(false)
}

// snippet-start:[lambda.rust.scenario.get_function]
/** Get the Lambda function with this Manager's name. */
pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
    info!("Getting lambda function");
    self.lambda_client
        .get_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await

```

```
        .map_err(anyhow::Error::from)
    }
    // snippet-end:[lambda.rust.scenario.get_function]

    // snippet-start:[lambda.rust.scenario.list_functions]
    /** List all Lambda functions in the current Region. */
    pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
        info!("Listing lambda functions");
        self.lambda_client
            .list_functions()
            .send()
            .await
            .map_err(anyhow::Error::from)
    }
    // snippet-end:[lambda.rust.scenario.list_functions]

    // snippet-start:[lambda.rust.scenario.invoke]
    /** Invoke the lambda function using calculator InvokeArgs. */
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }
    // snippet-end:[lambda.rust.scenario.invoke]

    // snippet-start:[lambda.rust.scenario.update_function_code]
    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;
```

```
        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }
    // snippet-end:[lambda.rust.scenario.update_function_code]

    // snippet-start:[lambda.rust.scenario.update_function_configuration]
    /** Update the environment for a function. */
    pub async fn update_function_configuration(
        &self,
        environment: Environment,
    ) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
        info!(
            ?environment,
            "Updating environment for {}", self.lambda_name
        );
        let updated = self
            .lambda_client
            .update_function_configuration()
            .function_name(self.lambda_name.clone())
            .environment(environment)
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(updated)
    }
    // snippet-end:[lambda.rust.scenario.update_function_configuration]

    // snippet-start:[lambda.rust.scenario.delete_function]
```

```
/** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
pub async fn delete_function(
    &self,
    location: Option<String>,
) -> (
    Result<DeleteFunctionOutput, anyhow::Error>,
    Result<DeleteRoleOutput, anyhow::Error>,
    Option<Result<DeleteObjectOutput, anyhow::Error>>,
) {
    info!("Deleting lambda function {}", self.lambda_name);
    let delete_function = self
        .lambda_client
        .delete_function()
        .function_name(self.lambda_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    info!("Deleting iam role {}", self.role_name);
    let delete_role = self
        .iam_client
        .delete_role()
        .role_name(self.role_name.clone())
        .send()
        .await
        .map_err(anyhow::Error::from);

    let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
        if let Some(location) = location {
            info!("Deleting object {location}");
            Some(
                self.s3_client
                    .delete_object()
                    .bucket(self.bucket.clone())
                    .key(location)
                    .send()
                    .await
                    .map_err(anyhow::Error::from),
            )
        } else {
            info!(?location, "Skipping delete object");
            None
        };
};
```

```

        (delete_function, delete_role, delete_object)
    }
    // snippet-end:[lambda.rust.scenario.delete_function]

    pub async fn cleanup(
        &self,
        location: Option<String>,
    ) -> (
        (
            Result<DeleteFunctionOutput, anyhow::Error>,
            Result<DeleteRoleOutput, anyhow::Error>,
            Option<Result<DeleteObjectOutput, anyhow::Error>>,
        ),
        Option<Result<DeleteBucketOutput, anyhow::Error>>,
    ) {
        let delete_function = self.delete_function(location).await;

        let delete_bucket = if self.own_bucket {
            info!("Deleting bucket {}", self.bucket);
            if delete_function.2.is_none() ||
delete_function.2.as_ref().unwrap().is_ok() {
                Some(
                    self.s3_client
                        .delete_bucket()
                        .bucket(self.bucket.clone())
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                None
            }
        } else {
            info!("No bucket to clean up");
            None
        };

        (delete_function, delete_bucket)
    }
}

/**

```



```

* Testing occurs primarily as an integration test running the `scenario` bin
successfully.
* Each action relies deeply on the internal workings and state of Amazon Simple
Storage Service (Amazon S3), Lambda, and IAM working together.
* It is therefore infeasible to mock the clients to test the individual actions.
*/
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's
    expected by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
    }
}

```

Un binario para ejecutar el escenario de principio a fin, con indicadores de líneas de comando para controlar algunos comportamientos. Este archivo es `src/bin/scenario.rs` en la caja.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction

```

## ## Scenario

A scenario runs at a command prompt and prints output to the user on the result of each service action. A scenario can run in one of two ways: straight through, printing out progress as it goes, or as an interactive question/answer script.

## ## Getting started with functions

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:

Note: Handlers don't use AWS SDK API calls.

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:

1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO, WARNING, ERROR).

It logs a few things at different levels, such as:

- \* DEBUG: Full event data.
- \* INFO: The calculation result.
- \* WARN~ING~: When a divide by zero error occurs.
- \* This will be the typical `RUST\_LOG` variable.

The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the following requirements:
  - \* Has an `assume_role` policy that grants 'lambda.amazonaws.com' the 'sts:AssumeRole' action.
  - \* Attaches the 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' managed role.
  - \* You must wait for ~10 seconds after the role is created before you can use it!
2. Create a function (CreateFunction) for the increment handler by packaging it as a zip and doing one of the following:
  - \* Adding it with CreateFunction Code.ZipFile.

- \* --or--
  - \* Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it with CreateFunction Code.S3Bucket/S3Key.
  - \* \_Note: Zipping the file does not have to be done in code.\_
  - \* If you have a waiter, use it to wait until the function is active. Otherwise, call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
  4. Update the function (UpdateFunctionCode) to the arithmetic handler by packaging it as a zip and doing one of the following:
    - \* Adding it with UpdateFunctionCode ZipFile.
    - \* --or--
    - \* Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/S3Key.
  5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or 'Failed').
  6. Update the environment variable by calling UpdateFunctionConfiguration and pass it a log level, such as:
    - \* Environment={'Variables': {'RUST\_LOG': 'TRACE'}}
  7. Invoke the function with an action from the list and a couple of values. Include LogType='Tail' to get logs in the result. Print the result of the calculation and the log.
  8. [Optional] Invoke the function to provoke a divide-by-zero error and show the log result.
  9. List all functions for the account, using pagination (ListFunctions).
  10. Delete the function (DeleteFunction).
  11. Delete the role.

Each step should use the function created in Service Actions to abstract calling the SDK.

```
*/

use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {
```

```
/// The AWS Region.
#[structopt(short, long)]
pub region: Option<String>,

// The bucket to use for the FunctionCode.
#[structopt(short, long)]
pub bucket: Option<String>,

// The name of the Lambda function.
#[structopt(short, long)]
pub lambda_name: Option<String>,

// The number to increment.
#[structopt(short, long, default_value = "12")]
pub inc: i32,

// The left operand.
#[structopt(long, default_value = "19")]
pub num_a: i32,

// The right operand.
#[structopt(long, default_value = "23")]
pub num_b: i32,

// The arithmetic operation.
#[structopt(short, long, default_value = "plus")]
pub operation: Operation,

#[structopt(long)]
pub cleanup: Option<bool>,

#[structopt(long)]
pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

// snippet-start:[lambda.rust.scenario.log_invoke_output]
fn log_invoke_output(invoker: &InvokeOutput, message: &str) {
    if let Some(payload) = invoker.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    }
}
```

```
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
// snippet-end:[lambda.rust.scenario.log_invoke_output]

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                ])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");

    let invoke = manager
```

```

        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with increased logging",
);

let invoke = manager
    .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
    .await?;
log_invoke_output(
    &invoke,
    "Invoked function configured as arithmetic with divide by zero",
);

Ok::<(), anyhow::Error>(( ))
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
opt.bucket.clone()).await;

    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };
};

```

```
    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    } else {
        let delete = manager.cleanup(key).await;
        info!(?delete, "Deleted function & cleaned up resources");
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Rust.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

## SAP ABAP

### SDK de SAP ABAP

#### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

```
TRY.
    "Create an AWS Identity and Access Management (IAM) role that grants AWS
    Lambda permission to write to logs."
    DATA(lv_policy_document) = `{` &&
        ` "Version": "2012-10-17", ` &&
        ` "Statement": [ ` &&
            `{ ` &&
                ` "Effect": "Allow", ` &&
```

```

        "Action": [ ` &&
            "sts:AssumeRole" ` &&
        ], ` &&
        "Principal": { ` &&
            "Service": [ ` &&
                "lambda.amazonaws.com" ` &&
            ] ` &&
        } ` &&
    } ` &&
] ` &&
} ` .

TRY.
    DATA(lo_create_role_output) = lo_iam->createrole(
        iv_rolename = iv_role_name
        iv_assumerolepolicydocument = lv_policy_document
        iv_description = 'Grant lambda permission to write to logs'
    ).
    MESSAGE 'IAM role created.' TYPE 'I'.
    WAIT UP TO 10 SECONDS.          " Make sure that the IAM role is
ready for use. "
    CATCH /aws1/cx_iamentityalrddyexex.
        MESSAGE 'IAM role already exists.' TYPE 'E'.
    CATCH /aws1/cx_iaminvalidinputex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_iammalformedplydocex.
        MESSAGE 'Policy document in the request is malformed.' TYPE 'E'.
ENDTRY.

TRY.
    lo_iam->attachrolepolicy(
        iv_rolename = iv_role_name
        iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole'
    ).
    MESSAGE 'Attached policy to the IAM role.' TYPE 'I'.
    CATCH /aws1/cx_iaminvalidinputex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
    CATCH /aws1/cx_iamplynotattachableex.
        MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
    CATCH /aws1/cx_iamunmodableentityex.

```



```

        MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
    ENDTRY.

    " Create a Lambda function and upload handler code. "
    " Lambda function performs 'increment' action on a number. "
    TRY.
        lo_lmd->createfunction(
            iv_functionname = iv_function_name
            iv_runtime = `python3.9`
            iv_role = lo_create_role_output->get_role( )->get_arn( )
            iv_handler = iv_handler
            io_code = io_initial_zip_file
            iv_description = 'AWS Lambda code example'
        ).
        MESSAGE 'Lambda function created.' TYPE 'I'.
    CATCH /aws1/cx_lmdcodestorageexcdex.
        MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
    CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
    CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
    ENDTRY.

    " Verify the function is in Active state "
    WHILE lo_lmd->getfunction( iv_functionname = iv_function_name )->
>get_configuration( )->ask_state( ) <> 'Active'.
        IF sy-index = 10.
            EXIT.                " Maximum 10 seconds. "
        ENDIF.
        WAIT UP TO 1 SECONDS.
    ENDWHILE.

    "Invoke the function with a single parameter and get results."
    TRY.
        DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
            `{` &&
            ` "action": "increment",` &&
            ` "number": 10` &&
            `}`
        ).
        DATA(lo_initial_invoke_output) = lo_lmd->invoke(
            iv_functionname = iv_function_name
            iv_payload = lv_json

```

```

        ).
        ov_initial_invoke_payload = lo_initial_invoke_output->get_payload( ).
        " ov_initial_invoke_payload is returned for testing purposes. "
        DATA(lo_writer_json) = cl_sxml_string_writer=>create( type =
if_sxml=>co_xt_json ).
        CALL TRANSFORMATION id SOURCE XML ov_initial_invoke_payload RESULT
XML lo_writer_json.
        DATA(lv_result) = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
        MESSAGE 'Lambda function invoked.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdinvrequestcontex.
        MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        CATCH /aws1/cx_lmdunsuppedmediatyp00.
        MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
        ENDTRY.

        " Update the function code and configure its Lambda environment with an
environment variable. "
        " Lambda function is updated to perform 'decrement' action also. "
        TRY.
            lo_lmd->updatefunctioncode(
                iv_functionname = iv_function_name
                iv_zipfile = io_updated_zip_file
            ).
            WAIT UP TO 10 SECONDS.          " Make sure that the update is
completed. "
            MESSAGE 'Lambda function code updated.' TYPE 'I'.
            CATCH /aws1/cx_lmdcodestorageexcdex.
            MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
            CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
            CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.

        TRY.
            DATA lt_variables TYPE /aws1/
cl_lmdenvironmentvaria00=>tt_environmentvariables.
            DATA ls_variable LIKE LINE OF lt_variables.

```

```

        ls_variable-key = 'LOG_LEVEL'.
        ls_variable-value = NEW /aws1/cl_lmdenvironmentvaria00( iv_value =
'info' ).
        INSERT ls_variable INTO TABLE lt_variables.

        lo_lmd->updatefunctionconfiguration(
            iv_functionname = iv_function_name
            io_environment = NEW /aws1/cl_lmdenvironment( it_variables =
lt_variables )
        ).
        WAIT UP TO 10 SECONDS.           " Make sure that the update is
completed. "
        MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in
progress.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.

        "Invoke the function with new parameters and get results. Display the
execution log that's returned from the invocation."
        TRY.
            lv_json = /aws1/cl_rt_util=>string_to_xstring(
                `{` &&
                `"action": "decrement",` &&
                `"number": 10` &&
                `}`
            ).
            DATA(lo_updated_invoke_output) = lo_lmd->invoke(
                iv_functionname = iv_function_name
                iv_payload = lv_json
            ).
            ov_updated_invoke_payload = lo_updated_invoke_output->get_payload( ).
            " ov_updated_invoke_payload is returned for testing purposes. "
            lo_writer_json = cl_sxml_string_writer=>create( type =
if_sxml=>co_xt_json ).
            CALL TRANSFORMATION id SOURCE XML ov_updated_invoke_payload RESULT
XML lo_writer_json.
            lv_result = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
            MESSAGE 'Lambda function invoked.' TYPE 'I'.

```

```

CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdinvrequestcontex.
    MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
CATCH /aws1/cx_lmdunsuppmediatyp00.
    MESSAGE 'Invoke request body does not have JSON as its content type.'
TYPE 'E'.
ENDTRY.

" List the functions for your account. "
TRY.
    DATA(lo_list_output) = lo_lmd->listfunctions( ).
    DATA(lt_functions) = lo_list_output->get_functions( ).
    MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
ENDTRY.

" Delete the Lambda function. "
TRY.
    lo_lmd->deletefunction( iv_functionname = iv_function_name ).
    MESSAGE 'Lambda function deleted.' TYPE 'I'.
CATCH /aws1/cx_lmdinvparamvalueex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_lmdresourcenotfoundex.
    MESSAGE 'The requested resource does not exist.' TYPE 'E'.
ENDTRY.

" Detach role policy. "
TRY.
    lo_iam->detachrolepolicy(
        iv_rolename = iv_role_name
        iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole'
    ).
    MESSAGE 'Detached policy from the IAM role.' TYPE 'I'.
CATCH /aws1/cx_iaminvalidinputex.
    MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
CATCH /aws1/cx_iamnosuchentityex.
    MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
CATCH /aws1/cx_iamplynnotattachableex.

```

```
        MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
    CATCH /aws1/cx_iamunmodableentityex.
        MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
    ENDTRY.

" Delete the IAM role. "
TRY.
    lo_iam->deleterole( iv_rolename = iv_role_name ).
    MESSAGE 'IAM role deleted.' TYPE 'I'.
    CATCH /aws1/cx_iamnosuchentityex.
        MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
    CATCH /aws1/cx_iamunmodableentityex.
        MESSAGE 'Service that depends on the service-linked role is not
modifiable.' TYPE 'E'.
    ENDTRY.

    CATCH /aws1/cx_rt_service_generic INTO lo_exception.
    DATA(lv_error) = lo_exception->get_longtext( ).
    MESSAGE lv_error TYPE 'E'.
ENDTRY.
```

- Para detalles acerca de la API, consulte los siguientes temas en la Referencia de la API del SDK de AWS para SAP ABAP.
  - [CreateFunction](#)
  - [DeleteFunction](#)
  - [GetFunction](#)
  - [Invoke](#)
  - [ListFunctions](#)
  - [UpdateFunctionCode](#)
  - [UpdateFunctionConfiguration](#)

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Escriba datos de actividad personalizados con una función de Lambda tras la autenticación de usuarios de Amazon Cognito mediante un AWS SDK

En el siguiente ejemplo de código, se muestra cómo escribir datos de actividad personalizados con una función de Lambda tras la autenticación de usuarios de Amazon Cognito.

- Utilice las funciones de administrador para añadir un usuario a un grupo de usuarios.
- Configure un grupo de usuarios para que llame a una función de Lambda para el desencadenador `PostAuthentication`.
- Inicie sesión con el nuevo usuario en Amazon Cognito.
- La función de Lambda escribe información personalizada en los registros de CloudWatch y en una tabla de DynamoDB.
- Obtenga y exhiba los datos personalizados de la tabla de DynamoDB y, a continuación, elimine los recursos.

Go

SDK para Go V2

### Note

Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el [Repositorio de ejemplos de código de AWS](#).

Ejecutar un escenario interactivo en un símbolo del sistema.

```
// ActivityLog separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type ActivityLog struct {
    helper          IScenarioHelper
    questioner     demotools.IQuestioner
    resources      Resources
    cognitoActor   *actions.CognitoActions
}
```

```
// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) ActivityLog {
    scenario := ActivityLog{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
    credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(userPoolId string, tableName string)
    (string, string) {
    log.Println("To facilitate this example, let's add a user to the user pool using
    administrator privileges.")
    users, err := runner.helper.GetKnownUsers(tableName)
    if err != nil {
        panic(err)
    }
    user := users.Users[0]
    log.Printf("Adding known user %v to the user pool.\n", user.UserName)
    err = runner.cognitoActor.AdminCreateUser(userPoolId, user.UserName,
    user.UserEmail)
    if err != nil {
        panic(err)
    }
    pwSet := false
    password := runner.questioner.AskPassword("\nEnter a password that has at least
    eight characters, uppercase, lowercase, numbers and symbols.\n"+
        "(the password will not display as you type):", 8)
    for !pwSet {
        log.Printf("\nSetting password for user '%v'.\n", user.UserName)
        err = runner.cognitoActor.AdminSetUserPassword(userPoolId, user.UserName,
        password)
        if err != nil {
            var invalidPassword *types.InvalidPasswordException
            if errors.As(err, &invalidPassword) {
                password = runner.questioner.AskPassword("\nEnter another password:", 8)
            } else {
```

```
        panic(err)
    }
    } else {
        pwSet = true
    }
}

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
// PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(userPoolId string,
activityLogArn string) {
    log.Println("Let's add a Lambda function to handle the PostAuthentication
trigger from Cognito.\n" +
        "This trigger happens after a user is authenticated, and lets your function
take action, such as logging\n" +
        "the outcome.")
    err := runner.cognitoActor.UpdateTriggers(
        userPoolId,
        actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
aws.String(activityLogArn)})
    if err != nil {
        panic(err)
    }
    runner.resources.triggers = append(runner.resources.triggers,
actions.PostAuthentication)
    log.Printf("Lambda function %v added to user pool %v to handle
PostAuthentication Cognito trigger.\n",
        activityLogArn, userPoolId)

    log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(clientId string, userName string, password
string) {
    log.Printf("Now we'll sign in user %v and check the results in the logs and the
DynamoDB table.", userName)
    runner.questioner.Ask("Press Enter when you're ready.")
    authResult, err := runner.cognitoActor.SignIn(clientId, userName, password)
```



```
    if err != nil {
        panic(err)
    }
    log.Println("Sign in successful.",
        "The PostAuthentication Lambda handler writes custom information to CloudWatch
        Logs.")

    runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
        *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
// table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(tableName string, userName
string) {
    log.Println("The PostAuthentication handler also writes login data to the
    DynamoDB table.")
    runner.questioner.Ask("Press Enter when you're ready to continue.")
    users, err := runner.helper.GetKnownUsers(tableName)
    if err != nil {
        panic(err)
    }
    for _, user := range users.Users {
        if user.UserName == userName {
            log.Println("The last login info for the user in the known users table is:")
            log.Printf("\t%+v", *user.LastLogin)
        }
    }
    log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *ActivityLog) Run(stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup()
        }
    }()

    log.Println(strings.Repeat("-", 88))
    log.Printf("Welcome\n")

    log.Println(strings.Repeat("-", 88))
}
```

```

stackOutputs, err := runner.helper.GetStackOutputs(stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(stackOutputs["TableName"])
userName, password := runner.AddUserToPool(stackOutputs["UserPoolId"],
stackOutputs["TableName"])

runner.AddActivityLogTrigger(stackOutputs["UserPoolId"],
stackOutputs["ActivityLogFunctionArn"])
runner.SignInUser(stackOutputs["UserPoolClientId"], userName, password)
runner.helper.ListRecentLogEvents(stackOutputs["ActivityLogFunction"])
runner.GetKnownUserLastLogin(stackOutputs["TableName"], userName)

runner.resources.Cleanup()

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

Controle el desencadenador PostAuthentication con una función de Lambda.

```

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
// format.
type LoginInfo struct {
    UserPoolId string `dynamodbav:"UserPoolId"`
    ClientId   string `dynamodbav:"ClientId"`
    Time      string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

```

```
LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := attributevalue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the PostAuthentication event by writing custom data to
// the logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
    event events.CognitoEventUserPoolsPostAuthentication)
    (events.CognitoEventUserPoolsPostAuthentication, error) {
    log.Printf("Received post authentication trigger from %v for user '%v'",
        event.TriggerSource, event.UserName)
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserName: event.UserName,
        UserEmail: event.Request.UserAttributes["email"],
        LastLogin: LoginInfo{
            UserPoolId: event.UserPoolID,
            ClientId: event.CallerContext.ClientID,
            Time: time.Now().Format(time.UnixDate),
        },
    },
}
// Write to CloudWatch Logs.
fmt.Printf("%#v", user)

// Also write to an external system. This examples uses DynamoDB to demonstrate.
userMap, err := attributevalue.MarshalMap(user)
if err != nil {
    log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
} else if len(userMap) == 0 {
    log.Printf("User info marshaled to an empty map.")
} else {
```

```

_, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
    Item:      userMap,
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
} else {
    log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
}
}

return event, nil
}

func main() {
    sdkConfig, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
}

```

Cree una estructura que lleve a cabo las tareas habituales.

```

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(stackName string) (actions.StackOutputs, error)
    PopulateUserTable(tableName string)
    GetKnownUsers(tableName string) (actions.UserList, error)
    AddKnownUser(tableName string, user actions.User)
    ListRecentLogEvents(functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.

```

```
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor *actions.CloudFormationActions
    cwlActor *actions.CloudWatchLogsActions
    isTestRun bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
dynamodb.NewFromConfig(sdkConfig)},
        cfnActor: &actions.CloudFormationActions{CfnClient:
cloudformation.NewFromConfig(sdkConfig)},
        cwlActor: &actions.CloudWatchLogsActions{CwlClient:
cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
structured format.
func (helper ScenarioHelper) GetStackOutputs(stackName string)
(actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(tableName string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(tableName)
    if err != nil {
        panic(err)
    }
}
```

```
    }
}

// GetKnownUsers gets the users from the known users table in a structured
// format.
func (helper ScenarioHelper) GetKnownUsers(tableName string) (actions.UserList,
error) {
    knownUsers, err := helper.dynamoActor.Scan(tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(tableName string, user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
        table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
        your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(functionName,
        *logStream.LogStreamName, 10)
    if err != nil {
        panic(err)
    }
}
```

```

for _, event := range events {
    log.Printf("\t%v", *event.Message)
}
log.Println(strings.Repeat("-", 88))
}

```

Cree una estructura que ajuste las acciones de Amazon Cognito.

```

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(userPoolId string,
    triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(context.TODO(),
    &cognitoidentityprovider.DescribeUserPoolInput{
        UserPoolId: aws.String(userPoolId),
    })
    if err != nil {

```

```

    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
userPoolId, err)
    return err
}
lambdaConfig := output.UserPool.LambdaConfig
for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
        lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
        lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
        lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
}
_, err = actor.CognitoClient.UpdateUserPool(context.TODO(),
&cognitoidentityprovider.UpdateUserPoolInput{
    UserPoolId:    aws.String(userPoolId),
    LambdaConfig: lambdaConfig,
})
if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(clientId string, userName string, password
string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(context.TODO(),
&cognitoidentityprovider.SignUpInput{
        ClientId: aws.String(clientId),
        Password: aws.String(password),
        Username: aws.String(userName),
        UserAttributes: []types.AttributeType{
            {Name: aws.String("email"), Value: aws.String(userEmail)},
        },
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {

```



```
    log.Println(*invalidPassword.Message)
} else {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
}
} else {
    confirmed = output.UserConfirmed
}
return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
authentication flow.
func (actor CognitoActions) SignIn(clientId string, userName string, password
string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(context.TODO(),
&cognitoidentityprovider.InitiateAuthInput{
    AuthFlow:      "USER_PASSWORD_AUTH",
    ClientId:      aws.String(clientId),
    AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
})
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    } else {
        authResult = output.AuthenticationResult
    }
    return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(clientId string, userName string)
(*types.CodeDeliveryDetailsType, error) {
```

```
output, err := actor.CognitoClient.ForgotPassword(context.TODO(),
&cognitoidentityprovider.ForgotPasswordInput{
    ClientId: aws.String(clientId),
    Username: aws.String(userName),
})
if err != nil {
    log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
userName, err)
}
return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
password.
func (actor CognitoActions) ConfirmForgotPassword(clientId string, code string,
userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(context.TODO(),
&cognitoidentityprovider.ConfirmForgotPasswordInput{
    ClientId:      aws.String(clientId),
    ConfirmationCode: aws.String(code),
    Password:      aws.String(password),
    Username:      aws.String(userName),
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
    }
}
return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(userAccessToken string) error {
    _, err := actor.CognitoClient.DeleteUser(context.TODO(),
&cognitoidentityprovider.DeleteUserInput{
    AccessToken: aws.String(userAccessToken),
})
}
```

```
    if err != nil {
        log.Printf("Couldn't delete user. Here's why: %v\n", err)
    }
    return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(userPoolId string, userName string,
    userEmail string) error {
    _, err := actor.CognitoClient.AdminCreateUser(context.TODO(),
        &cognitoidentityprovider.AdminCreateUserInput{
            UserPoolId:    aws.String(userPoolId),
            Username:      aws.String(userName),
            MessageAction: types.MessageActionTypeSuppress,
            UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
                aws.String(userEmail)}}},
        })
    if err != nil {
        var userExists *types.UsernameExistsException
        if errors.As(err, &userExists) {
            log.Printf("User %v already exists in the user pool.", userName)
            err = nil
        } else {
            log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
        }
    }
    return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(userPoolId string, userName
    string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(context.TODO(),
        &cognitoidentityprovider.AdminSetUserPasswordInput{
            Password:    aws.String(password),
            UserPoolId: aws.String(userPoolId),
```

```

Username:  aws.String(userName),
Permanent: true,
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
    }
}
return err
}

```

Cree una estructura que ajuste las acciones de DynamoDB.

```

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
    UserName string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId string
    Time string
}

// UserList defines a list of users.
type UserList struct {

```

```
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
// strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(tableName string) error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
        item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
        if err != nil {
            log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
err)
            return err
        }
        writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
&types.PutRequest{Item: item}})
    }
    _, err = actor.DynamoClient.BatchWriteItem(context.TODO(),
&dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
    if err != nil {
        log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
tableName, err)
    }
    return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(tableName string) (UserList, error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(context.TODO(), &dynamodb.ScanInput{
```

```

    TableName: aws.String(tableName),
  })
  if err != nil {
    log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
err)
  } else {
    err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
    if err != nil {
      log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
    }
  }
  return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(tableName string, user User) error {
  userItem, err := attributevalue.MarshalMap(user)
  if err != nil {
    log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
  }
  _, err = actor.DynamoClient.PutItem(context.TODO(), &dynamodb.PutItemInput{
    Item:      userItem,
    TableName: aws.String(tableName),
  })
  if err != nil {
    log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
  }
  return err
}

```

Cree una estructura que ajuste las acciones de los registros de CloudWatch.

```

type CloudWatchLogsActions struct {
  CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(functionName string)
(types.LogStream, error) {
  var logStream types.LogStream

```

```

logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.DescribeLogStreams(context.TODO(),
&cloudwatchlogs.DescribeLogStreamsInput{
    Descending:  aws.Bool(true),
    Limit:       aws.Int32(1),
    LogGroupName: aws.String(logGroupName),
    OrderBy:    types.OrderByLastEventTime,
})
if err != nil {
    log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
logGroupName, err)
} else {
    logStream = output.LogStreams[0]
}
return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
stream.
func (actor CloudWatchLogsActions) GetLogEvents(functionName string,
logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(context.TODO(),
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
logStreamName, err)
} else {
    events = output.Events
}
return events, err
}

```

Cree una estructura que ajuste las acciones de AWS CloudFormation.

```

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(stackName string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(context.TODO(),
        &cloudformation.DescribeStacksInput{
            StackName: aws.String(stackName),
        })
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
            stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}

```

## Eliminación de recursos.

```

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}

```



```
func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup() {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()

    wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
        "during this demo (y/n)?", "y")
    if wantDelete {
        for _, accessToken := range resources.userAccessTokens {
            err := resources.cognitoActor.DeleteUser(accessToken)
            if err != nil {
                log.Println("Couldn't delete user during cleanup.")
                panic(err)
            }
            log.Println("Deleted user.")
        }
        triggerList := make([]actions.TriggerInfo, len(resources.triggers))
        for i := 0; i < len(resources.triggers); i++ {
            triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
        }
        err := resources.cognitoActor.UpdateTriggers(resources.userPoolId,
triggerList...)
        if err != nil {
            log.Println("Couldn't update Cognito triggers during cleanup.")
            panic(err)
        }
        log.Println("Removed Cognito triggers from user pool.")
    } else {
```

```
log.Println("Be sure to remove resources when you're done with them to avoid
unexpected charges!")
}
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Go.
  - [AdminCreateUser](#)
  - [AdminSetUserPassword](#)
  - [DeleteUser](#)
  - [InitiateAuth](#)
  - [UpdateUserPool](#)

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Ejemplos sin servidor para Lambda que utilizan SDK de AWS

Los siguientes ejemplos de código muestran cómo utilizar Lambda con los SDK de AWS.

### Ejemplos

- [Conexión a una base de datos de Amazon RDS en una función de Lambda](#)
- [Invocar una función de Lambda desde un desencadenador de Kinesis](#)
- [Invocación de una función de Lambda desde un desencadenador de DynamoDB](#)
- [Invocación de una función de Lambda desde un desencadenador de Amazon DocumentDB](#)
- [Invocación de una función de Lambda desde un desencadenador de Amazon MSK](#)
- [Invocación de una función de Lambda desde un desencadenador de Amazon S3](#)
- [Invocar una función de Lambda desde un desencadenador de Amazon SNS](#)
- [Invocar una función de Lambda desde un desencadenador de Amazon SQS](#)
- [Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis](#)

- [Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB](#)
- [Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.](#)

## Conexión a una base de datos de Amazon RDS en una función de Lambda

En el siguiente ejemplo de código, se muestra cómo implementar una función de Lambda que se conecta a una base de datos de RDS. La función realiza una solicitud sencilla a la base de datos y devuelve el resultado.

Go

SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Golang v2 code here.
*/

package main

import (
    "context"
    "database/sql"
    "encoding/json"
    "fmt"
    "os"

    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"

```

```
"github.com/aws/aws-sdk-go-v2/feature/rds/auth"
_ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

    var dbName string = os.Getenv("DatabaseName")
    var dbUser string = os.Getenv("DatabaseUser")
    var dbHost string = os.Getenv("DBHost") // Add hostname without https
    var dbPort int = os.Getenv("Port")      // Add port number
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = os.Getenv("AWS_REGION")

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
        context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
        panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    defer db.Close()

    var sum int
    err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
    if err != nil {
        panic(err)
    }
}
```

```
s := fmt.Sprintf(sum)
message := fmt.Sprintf("The selected sum is: %s", s)

messageBytes, err := json.Marshal(message)
if err != nil {
    return nil, err
}

messageString := string(messageBytes)
return map[string]interface{}{
    "statusCode": 200,
    "headers":    map[string]string{"Content-Type": "application/json"},
    "body":       messageString,
}, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
```

```
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
    RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
        event, Context context) {
        APIGatewayProxyResponseEvent response = new
        APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
            useSSL=true&requireSSL=true",
                System.getenv("ProxyHostName"),
                System.getenv("Port"),
                System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
                DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
                token);
                PreparedStatement statement =
                connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }
        }
    }
}
```

```
    }

    } catch (Exception e) {
        response.setStatusCode(500);
        response.setBody("Error: " + e.getMessage());
    }

    return response;
}

private String createAuthToken() {
    // Create RDS Data Service client
    RdsDataClient rdsDataClient = RdsDataClient.builder()
        .region(Region.of(System.getenv("AWS_REGION")))
        .credentialsProvider(DefaultCredentialsProvider.create())
        .build();

    // Define authentication request
    ExecuteStatementRequest request = ExecuteStatementRequest.builder()
        .resourceArn(System.getenv("ProxyHostName"))
        .secretArn(System.getenv("DBUserName"))
        .database(System.getenv("DBName"))
        .sql("SELECT 'RDS IAM Authentication'")
        .build();

    // Execute request and obtain authentication token
    ExecuteStatementResponse response =
rdsDataClient.executeStatement(request);
    Field tokenField = response.records().get(0).get(0);

    return tokenField.stringValue();
}
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {
```



```
// Obtain auth token
const token = await createAuthToken();
// Define connection configuration
let connectionConfig = {
  host: process.env.ProxyHostName,
  user: process.env.DBUserName,
  password: token,
  database: process.env.DBName,
  ssl: 'Amazon RDS'
}
// Create the connection to the DB
const conn = await mysql.createConnection(connectionConfig);
// Obtain the result of the query
const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
return res;
}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
// the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
```

```
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

  // Create RDS Signer object
  const signer = new Signer({
    hostname: proxy_host_name,
    port: port,
    region: aws_region,
    username: db_user_name
  });

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
  try {
    // Obtain auth token
    const token = await createAuthToken();
    const conn = await mysql.createConnection({
      host: proxy_host_name,
      user: db_user_name,
      password: token,
      database: db_name,
      ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
  }
  catch (err) {
    console.log(err);
  }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
  body: string }> => {
  // Execute database flow
```

```
const result = await dbOps();

// Return error if result is undefined
if (result == undefined)
    return {
        statusCode: 500,
        body: JSON.stringify(`Error with connection to DB host`)
    }

// Return result
return {
    statusCode: 200,
    body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
};
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';
```

```
class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
        AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(
```

```

"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
    $connectionConfig['user'],
    $connectionConfig['password'],
    [
        PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
        PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
    ]
);

// Obtain the result of the query
$stmt = $conn->prepare('SELECT ?+? AS sum');
$stmt->execute([3, 2]);

return $stmt->fetch(PDO::FETCH_ASSOC);
}

/**
 * @param mixed $event
 * @param Context $context
 * @return array
 */
public function handle(mixed $event, Context $context): array
{
    $this->logger->info("Processing query");

    // Execute database flow
    $result = $this->getQueryResults();

    return [
        'sum' => $result['sum']
    ];
}
}

$logger = new StderrLogger();
return new Handler($logger);

```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']

# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
```

```
        db=db_name,
        port=port,
        ssl={'ca': 'Amazon RDS'} # Ensure you have the CA bundle for SSL
connection
    )

    with connection.cursor() as cursor:
        cursor.execute('SELECT %s + %s AS sum', (3, 2))
        result = cursor.fetchone()

    return result

except Exception as e:
    return (f"Error: {str(e)}") # Return an error message if an exception
occurs
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
    endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
    port = ENV['Port']           # 3306
    user = ENV['DBUser']
    region = ENV['DBRegion']     # 'us-east-1'
```

```
db_name = ENV['DBName']

credentials = Aws::Credentials.new(
  ENV['AWS_ACCESS_KEY_ID'],
  ENV['AWS_SECRET_ACCESS_KEY'],
  ENV['AWS_SESSION_TOKEN']
)

rds_client = Aws::RDS::AuthTokenGenerator.new(
  region: region,
  credentials: credentials
)

token = rds_client.auth_token(
  endpoint: endpoint+ ':' + port,
  user_name: user,
  region: region
)

begin
  conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/var/task/global-bundle.pem',
    sslverify: true,
    enable_clear_text_plugin: true
  )
  a = 3
  b = 2
  result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
  puts result
  conn.close
  {
    statusCode: 200,
    body: result.to_json
  }
rescue => e
  puts "Database connection failed due to #{e}"
end
end
```



## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Conexión a una base de datos de Amazon RDS en una función de Lambda mediante Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
```

```

    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
SignableBody::Bytes(&[]))
        .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");

```

```
let db_port = env::var("DB_PORT")
    .expect("DB_PORT must be set")
    .parse::<u16>()
    .expect("PORT must be a valid number");
let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

let opts = PgConnectOptions::new()
    .host(&db_host)
    .port(db_port)
    .username(&db_user_name)
    .password(&token)
    .database(&db_name)
    .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
    .ssl_mode(sqlx::postgres::PgSslMode::Require);

let pool = sqlx::postgres::PgPoolOptions::new()
    .connect_with(opts)
    .await?;

let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
    .bind(3)
    .bind(2)
    .fetch_one(&pool)
    .await?;

println!("Result: {:?}", result);

Ok(json!({
    "statusCode": 200,
    "content-type": "text/plain",
    "body": format!("The selected sum is: {result}")
}))
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocar una función de Lambda desde un desencadenador de Kinesis

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una transmisión de Kinesis. La función recupera la carga útil de Kinesis, la decodifica desde Base64 y registra el contenido del registro.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de Kinesis con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
        }
    }
}
```


```
        return;
    }

    foreach (var record in evnt.Records)
    {
        try
        {
            Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
            string data = await GetRecordDataAsync(record.Kinesis, context);
            Logger.LogInformation($"Data: {data}");
            // TODO: Do interesting work based on the new data
        }
        catch (Exception ex)
        {
            Logger.LogError($"An error occurred {ex.Message}");
            throw;
        }
    }
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}
```

Go

SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de Kinesis con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
    if len(kinesisEvent.Records) == 0 {
        log.Printf("empty Kinesis event received")
        return nil
    }

    for _, record := range kinesisEvent.Records {
        log.Printf("processed Kinesis event with EventId: %v", record.EventID)
        recordDataBytes := record.Kinesis.Data
        recordDataText := string(recordDataBytes)
        log.Printf("record data: %v", recordDataText)
        // TODO: Do interesting work based on the new data
    }
    log.Printf("successfully processed %v records", len(kinesisEvent.Records))
    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
records");
    }
}
```

```
        return null;
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    for (const record of event.Records) {
        try {
            console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
            const recordData = await getRecordDataAsync(record.kinesis);
            console.log(`Record Data: ${recordData}`);
            // TODO: Do interesting work based on the new data
        } catch (err) {
            console.error(`An error occurred ${err}`);
            throw err;
        }
    }
    console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}
```



## Uso de un evento de Kinesis con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
    }
}
```

```
$records = $event->getRecords();
foreach ($records as $record) {
    $data = $record->getData();
    $this->logger->info(json_encode($data));
    // TODO: Do interesting work based on the new data

    // Any exception thrown will be logged and the invocation will be
marked as failed
}
$totalRecords = count($records);
$this->logger->info("Successfully processed $totalRecords records");
}
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Kinesis con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
```

```
    except Exception as e:
        print(f"An error occurred {e}")
        raise e
print(f"Successfully processed {len(event['Records'])} records.")
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de Kinesis con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
  here.
```

```
    return data
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Kinesis con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
{
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
{}", record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    })
}
```

```
});

tracing::info!(
    "Successfully processed {} records",
    event.payload.records.len()
);

Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocación de una función de Lambda desde un desencadenador de DynamoDB

En los siguientes ejemplos de código se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una secuencia de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

## .NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
    }
}
```

```
}  
}
```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de DynamoDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
package main  
  
import (  
    "context"  
    "github.com/aws/aws-lambda-go/lambda"  
    "github.com/aws/aws-lambda-go/events"  
    "fmt"  
)  
  
func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,  
error) {  
    if len(event.Records) == 0 {  
        return nil, fmt.Errorf("received empty event")  
    }  
  
    for _, record := range event.Records {  
        LogDynamoDBRecord(record)  
    }  
  
    message := fmt.Sprintf("Records processed: %d", len(event.Records))  
    return &message, nil  
}  
  
func main() {
```



```
lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
    fmt.Println(record.EventID)
    fmt.Println(record.EventName)
    fmt.Printf("%+v\n", record.Change)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
    com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
        GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```

        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
    GSON.toJson(record.getDynamodb()));
    }
}

```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante JavaScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};

```

### Consumo de un evento de DynamoDB con Lambda mediante TypeScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
};

```

```
event.Records.forEach(record => {
    logDynamoDBRecord(record);
});
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }
}
```

```
}

/**
 * @throws JsonException
 * @throws \Bref\Event\InvalidLambdaEvent
 */
public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
{
    $this->logger->info("Processing DynamoDb table items");
    $records = $event->getRecords();

    foreach ($records as $record) {
        $eventName = $record->getEventName();
        $keys = $record->getKeys();
        $old = $record->getOldImage();
        $new = $record->getNewImage();

        $this->logger->info("Event Name:". $eventName. "\n");
        $this->logger->info("Keys:". json_encode($keys). "\n");
        $this->logger->info("Old Image:". json_encode($old). "\n");
        $this->logger->info("New Image:". json_encode($new));

        // TODO: Do interesting work based on the new data

        // Any exception thrown will be logged and the invocation will be
        marked as failed
    }

    $totalRecords = count($records);
    $this->logger->info("Successfully processed $totalRecords items");
}
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event:, context:)
  return 'received empty event' if event['Records'].empty?

  event['Records'].each do |record|
    log_dynamodb_record(record)
  end

  "Records processed: #{event['Records'].length}"
end

def log_dynamodb_record(record)
  puts record['eventID']
  puts record['eventName']
  puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de DynamoDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
  event::dynamodb::{Event, EventRecord},
};
```

```
// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
  ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())
}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();
}
```

```
let func = service_fn(function_handler);
lambda_runtime::run(func).await?;
Ok(())
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocación de una función de Lambda desde un desencadenador de Amazon DocumentDB

En los siguientes ejemplos de código se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una secuencia de cambios de DocumentDB. La función recupera la carga útil de DocumentDB y registra el contenido del registro.

Go

SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumir un evento de Amazon DocumentDB con Lambda utilizando Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

package main

import (
    "context"
    "encoding/json"
    "fmt"
```



```
"github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
    Events []Record `json:"events"`
}

type Record struct {
    Event struct {
        OperationType string `json:"operationType"`
        NS             struct {
            DB   string `json:"db"`
            Coll string `json:"coll"`
        } `json:"ns"`
        FullDocument interface{} `json:"fullDocument"`
    } `json:"event"`
}

func main() {
    lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
    fmt.Println("Loading function")
    for _, record := range event.Events {
        logDocumentDBEvent(record)
    }

    return "OK", nil
}

func logDocumentDBEvent(record Record) {
    fmt.Printf("Operation type: %s\n", record.Event.OperationType)
    fmt.Printf("db: %s\n", record.Event.NS.DB)
    fmt.Printf("collection: %s\n", record.Event.NS.Coll)
    docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", " ")
    fmt.Printf("Full document: %s\n", string(docBytes))
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de Amazon DocumentDB con Lambda mediante JavaScript.

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

### Consumo de un evento de Amazon DocumentDB con Lambda mediante TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
```

```
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de Amazon DocumentDB con Lambda mediante PHP.

```
<?php

require __DIR__.'./vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {
        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
    }
}
```

```
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
        "\n";
    }
}
return new DocumentDBEventHandler();
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Consumo de un evento de Amazon DocumentDB con Lambda mediante Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'
```

```
def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de Amazon DocumentDB con Lambda mediante Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
```

```
puts "collection: #{collection}"
puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de Amazon DocumentDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
};

// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;
```

```
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())
}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}" , record.event);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocación de una función de Lambda desde un desencadenador de Amazon MSK

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de un clúster de Amazon MSK. La función recupera la carga útil de MSK y registra el contenido del registro.

Java

SDK para Java 2.x

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de Amazon MSK con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);
            }
        }
    }
}
```



```
        byte[] value = Base64.getDecoder().decode(record.getValue());
        String message = new String(value);
        System.out.println("Message: " + message);
    }
}

return null;
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de Amazon MSK con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
        console.log('Key: ', key)
        // Iterate through records
        event.records[key].map((record) => {
            console.log('Record: ', record)
            // Decode base64
            const msg = Buffer.from(record.value, 'base64').toString()
            console.log('Message:', msg)
        })
    }
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocación de una función de Lambda desde un desencadenador de Amazon S3

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al cargar un objeto en un bucket de S3. La función recupera el nombre del bucket de S3 y la clave del objeto del parámetro de evento y llama a la API de Amazon S3 para recuperar y registrar el tipo de contenido del objeto.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de S3 con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace S3Integration
{
    public class Function
```

```
{
    private static AmazonS3Client _s3Client;
    public Function() : this(null)
    {
    }

    internal Function(AmazonS3Client s3Client)
    {
        _s3Client = s3Client ?? new AmazonS3Client();
    }

    public async Task<string> Handler(S3Event evt, ILambdaContext context)
    {
        try
        {
            if (evt.Records.Count <= 0)
            {
                context.Logger.LogLine("Empty S3 Event received");
                return string.Empty;
            }

            var bucket = evt.Records[0].S3.Bucket.Name;
            var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

            context.Logger.LogLine($"Request is for {bucket} and {key}");

            var objectResult = await _s3Client.GetObjectAsync(bucket, key);


            context.Logger.LogLine($"Returning {objectResult.Key}");

            return objectResult.Key;
        }
        catch (Exception e)
        {
            context.Logger.LogLine($"Error processing request -
{e.Message}");

            return string.Empty;
        }
    }
}
```

## Go

## SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de S3 con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "log"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Printf("failed to load default config: %s", err)
        return err
    }
    s3Client := s3.NewFromConfig(sdkConfig)

    for _, record := range s3Event.Records {
        bucket := record.S3.Bucket.Name
        key := record.S3.Object.URLDecodedKey
        headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
            Bucket: &bucket,
            Key:    &key,
        })
        if err != nil {
            log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
        }
    }
}
```

```
    return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
*headOutput.ContentType)
}

return nil
}

func main() {
  lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
  com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNotifi

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);
            String srcBucket = record.getS3().getBucket().getName();
            String srcKey = record.getS3().getObject().getUrlDecodedKey();

            S3Client s3Client = S3Client.builder().build();
            HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
srcKey);

            logger.info("Successfully retrieved " + srcBucket + "/" + srcKey + " of
type " + headObject.contentType());

            return "Ok";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String bucket,
String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
            .bucket(bucket)
            .key(key)
            .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de S3 con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

exports.handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g,
  ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
    sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

## Uso de un evento de S3 con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });
```

```
export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
    they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de S3 con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;
```



```
require __DIR__ . '/vendor/autoload.php';

class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
                $bucket . '. Make sure they exist and your bucket is in the same region as this
                function.' . "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')

def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],
    encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and
        your bucket is in the same region as this function.'.format(key, bucket))
        raise e
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Consumo de un evento de S3 con Lambda mediante Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']['key'],
  Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they exist
    and your bucket is in the same region as this function."
    raise e
  end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de S3 con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::Client;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
```

```
tracing::info!(records = ?evt.payload.records.len(), "Received request from
SQS");

if evt.payload.records.len() == 0 {
    tracing::info!("Empty S3 event received");
}

let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket
name to exist");
let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
exist");

tracing::info!("Request is for {} and object {}", bucket, key);

let s3_get_object_result = s3_client
    .get_object()
    .bucket(bucket)
    .key(key)
    .send()
    .await;

match s3_get_object_result {
    Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
contains a 'body' property of type ByteStream"),
    Err(_) => tracing::info!("Failure with S3 Get Object request")
}

Ok(())
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocar una función de Lambda desde un desencadenador de Amazon SNS

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir mensajes de un tema de SNS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

## .NET

### AWS SDK for .NET

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
    ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
            {record.Sns.Message}");
        }
    }
}
```

```
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        processMessage(record)
    }
}
```

```
    fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
    message := record.SNS.Message
    fmt.Printf("Processed message: %s\n", message)
    // TODO: Process your record here
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;

import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;
```



```
@Override
public Boolean handleRequest(SNSEvent event, Context context) {
    logger = context.getLogger();
    List<SNSRecord> records = event.getRecords();
    if (!records.isEmpty()) {
        Iterator<SNSRecord> recordsIter = records.iterator();
        while (recordsIter.hasNext()) {
            processRecord(recordsIter.next());
        }
    }
    return Boolean.TRUE;
}

public void processRecord(SNSRecord record) {
    try {
        String message = record.getSNS().getMessage();
        logger.log("message: " + message);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

### Uso de un evento de SNS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
  }
}
```

```
        throw err;
    }
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
```

```
public function handleSns(SnsEvent $event, Context $context): void
{
    foreach ($event->getRecords() as $record) {
        $message = $record->getMessage();

        // TODO: Implement your custom processing logic here
        // Any exception thrown will be logged and the invocation will be
        marked as failed

        echo "Processed Message: $message" . PHP_EOL;
    }
}

return new Handler();
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here
```

```
except Exception as e:
    print("An error occurred")
    raise e
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SNS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
  rescue StandardError => e
    puts("Error processing message: #{e}")
    raise
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SNS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features
// = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features =
// ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for record in event.payload.records {
        process_record(&record)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocar una función de Lambda desde un desencadenador de Amazon SQS

En los siguientes ejemplos de código, se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir mensajes de una cola de SQS. La función recupera los mensajes del parámetro de eventos y registra el contenido de cada mensaje.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SQS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }


    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
        ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
            Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```



## Go

## SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Uso de un evento de SQS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
    for _, record := range event.Records {
        err := processMessage(record)
        if err != nil {
            return err
        }
    }
    fmt.Println("done")
    return nil
}

func processMessage(record events.SQSMessage) error {
    fmt.Printf("Processed message %s\n", record.Body)
    // TODO: Do interesting work based on the new message
    return nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
    }
}
```

```
}  
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
exports.handler = async (event, context) => {  
  for (const message of event.Records) {  
    await processMessageAsync(message);  
  }  
  console.info("done");  
};  
  
async function processMessageAsync(message) {  
  try {  
    console.log(`Processed message ${message.body}`);  
    // TODO: Do interesting work based on the new message  
    await Promise.resolve(1); //Placeholder for actual async work  
  } catch (err) {  
    console.error("An error occurred");  
    throw err;  
  }  
}
```

### Uso de un evento de SQS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Uso de un evento de SQS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].each do |message|
        process_message(message)
    end
    puts "done"
end

def process_message(message)
    begin
        puts "Processed message #{message['body']}"
        # TODO: Do interesting work based on the new message
    end
end
```

```
rescue StandardError => err
  puts "An error occurred"
  raise err
end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Uso de un evento de SQS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
            record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
        time.
        .without_time()
}
```

```
        .init();

        run(service_fn(function_handler)).await
    }
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Kinesis

En los siguientes ejemplos de código, se muestra cómo implementar una respuesta parcial por lotes para las funciones de Lambda que reciben eventos de una transmisión de Kinesis. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
```



```

[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
        ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
                item immediately.
                Lambda will immediately begin to retry processing from this
                failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
                    List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
                        { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                }
            }
        }
    }
}

```

```

        };
    }
}
    Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
    return new StreamsEventResponse();
}

private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
ILambdaContext context)
{
    byte[] bytes = record.Data.ToArray();
    string data = Encoding.UTF8.GetString(bytes);
    await Task.CompletedTask; //Placeholder for actual async work
    return data;
}
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
}

```

Go

## SDK para Go V2

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, record := range kinesisEvent.Records {
        curRecordSequenceNumber := ""

        // Process your record
        if /* Your record processing condition here */ {
            curRecordSequenceNumber = record.Kinesis.SequenceNumber
        }

        // Add a condition to check if the record processing failed
        if curRecordSequenceNumber != "" {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": curRecordSequenceNumber
            })
        }
    }

    kinesisBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return kinesisBatchResponse, nil
}

func main() {
    lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```

        /* Since we are working with streams, we can return the failed
        item immediately.
           Lambda will immediately begin to retry processing from this
        failed item onwards. */
        batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
        return new StreamsEventResponse(batchItemFailures);
    }
}

return new StreamsEventResponse(batchItemFailures);
}
}

```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Javascript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.

```

```

        Lambda will immediately begin to retry processing from this failed
        item onwards. */
        return {
            batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
        };
    }
}
console.log(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}

```

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante TypeScript.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    KinesisStreamEvent,
    Context,
    KinesisStreamHandler,
    KinesisStreamRecordPayload,
    KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
    logLevel: "INFO",
    serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
    event: KinesisStreamEvent,
    context: Context
): Promise<KinesisStreamBatchResponse> => {
    for (const record of event.Records) {

```

```
try {
  logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
  const recordData = await getRecordDataAsync(record.kinesis);
  logger.info(`Record Data: ${recordData}`);
  // TODO: Do interesting work based on the new data
} catch (err) {
  logger.error(`An error occurred ${err}`);
  /* Since we are working with streams, we can return the failed item
  immediately.
     Lambda will immediately begin to retry processing from this failed
  item onwards. */
  return {
    batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
  };
}
logger.info(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}
```



```
// change format for the response
$failures = array_map(
    fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
    $failedRecords
);

return [
    'batchItemFailures' => $failures
];
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
```

```

        return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}

```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Ruby.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
      # immediately.
      # Lambda will immediately begin to retry processing from this failed item
      # onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end
end

```

```

puts "Successfully processed #{event['Records'].length} records."
{ batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end

```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de Kinesis con Lambda mediante Rust.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
    Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }
}

```

```

for record in &event.payload.records {
    tracing::info!(
        "EventId: {}",
        record.event_id.as_deref().unwrap_or_default()
    );

    let record_processing_result = process_record(record);

    if record_processing_result.is_err() {
        response.batch_item_failures.push(KinesisBatchItemFailure {
            item_identifier: record.kinesis.sequence_number.clone(),
        });
        /* Since we are working with streams, we can return the failed item
immediately.
        Lambda will immediately begin to retry processing from this failed
item onwards. */
        return Ok(response);
    }
}

tracing::info!(
    "Successfully processed {} records",
    event.payload.records.len()
);

Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

En los siguientes ejemplos de código se muestra cómo implementar una respuesta parcial por lotes para las funciones de Lambda que reciben eventos de una secuencia de DynamoDB. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
        ILambdaContext context)
    {
        context.Logger.LogInformation($"Beginning to process
        {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
                { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
```

```

        streamsEventResponse.BatchItemFailures = batchItemFailures;
    }

    context.Logger.LogInformation("Stream processing complete.");
    return streamsEventResponse;
}
}

```

## Go

### SDK para Go V2

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Go.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
    ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
    BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
(*BatchResult, error) {
    var batchItemFailures []BatchItemFailure
    curRecordSequenceNumber := ""

```

```
for _, record := range event.Records {
    // Process your record
    curRecordSequenceNumber = record.Change.SequenceNumber
}

if curRecordSequenceNumber != "" {
    batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
curRecordSequenceNumber})
}

batchResult := BatchResult{
    BatchItemFailures: batchItemFailures,
}

return &batchResult, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
```



```
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
    Serializable> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
input.getRecords()) {
            try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
item immediately.
                Lambda will immediately begin to retry processing from this
failed item onwards. */
                batchItemFailures.add(new
StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

        return new StreamsEventResponse();
    }
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

### Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
```

```
DynamoDBStreamEvent,  
} from "aws-lambda";  
  
export const handler = async (  
  event: DynamoDBStreamEvent  
) : Promise<DynamoDBBatchResponse> => {  
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];  
  let curRecordSequenceNumber;  
  
  for (const record of event.Records) {  
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;  
  
    if (curRecordSequenceNumber) {  
      batchItemFailures.push({  
        itemIdentifier: curRecordSequenceNumber,  
      });  
    }  
  }  
  
  return { batchItemFailures: batchItemFailures };  
};
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante PHP.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
<?php  
  
# using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
```

```
        $failedRecords
    );

    return [
        'batchItemFailures' => $failures
    ];
}

$logger = new StderrLogger();
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
curRecordSequenceNumber}]}
```

```
return {"batchItemFailures":[]}
```

## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  records = event["Records"]
  cur_record_sequence_number = ""

  records.each do |record|
    begin
      # Process your record
      cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
      # Return failed record's sequence number
      return {"batchItemFailures" => [{"itemIdentifier" =>
cur_record_sequence_number}]}
    end
  end

  {"batchItemFailures" => []}
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

### Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
    Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
    }
}
```

```

    return Ok(response);
}

for record in records {
    tracing::info!("EventId: {}", record.event_id);

    // Couldn't find a sequence number
    if record.change.sequence_number.is_none() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: Some("").to_string(),
        });
        return Ok(response);
    }

    // Process your record here...
    if process_record(record).is_err() {
        response.batch_item_failures.push(DynamoDbBatchItemFailure {
            item_identifier: record.change.sequence_number.clone(),
        });
        /* Since we are working with streams, we can return the failed item
immediately.
        Lambda will immediately begin to retry processing from this failed
item onwards. */
        return Ok(response);
    }
}

tracing::info!("Successfully processed {} record(s)", records.len());

Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}

```



```
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Notificación de los errores de los elementos del lote de las funciones de Lambda mediante un desencadenador de Amazon SQS.

En los siguientes ejemplos de código, se muestra cómo implementar una respuesta parcial por lotes para las funciones de Lambda que reciben eventos de una cola de SQS. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

.NET

AWS SDK for .NET

### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]
namespace sqsSample;


public class Function
{
```

```
public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
ILambdaContext context)
{
    List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
List<SQSBatchResponse.BatchItemFailure>();
    foreach(var message in evnt.Records)
    {
        try
        {
            //process your message
            await ProcessMessageAsync(message, context);
        }
        catch (System.Exception)
        {
            //Add failed message identifier to the batchItemFailures list
            batchItemFailures.Add(new
SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
        }
    }
    return new SQSBatchResponse(batchItemFailures);
}

private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
ILambdaContext context)
{
    if (String.IsNullOrEmpty(message.Body))
    {
        throw new Exception("No Body in SQS Message.");
    }
    context.Logger.LogInformation($"Processed message {message.Body}");
    // TODO: Do interesting work based on the new message
    await Task.CompletedTask;
}
}
```

## Go

## SDK para Go V2

 Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
    (map[string]interface{}, error) {
    batchItemFailures := []map[string]interface{}{}

    for _, message := range sqsEvent.Records {

        if /* Your message processing condition here */ {
            batchItemFailures = append(batchItemFailures, map[string]interface{}{
                "itemIdentifier": message.MessageId})
        }
    }

    sqsBatchResponse := map[string]interface{}{
        "batchItemFailures": batchItemFailures,
    }
    return sqsBatchResponse, nil
}

func main() {
```

```
lambda.Start(handler)
}
```

## Java

### SDK para Java 2.x

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {

        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
ArrayList<SQSBatchResponse.BatchItemFailure>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
                messageId = message.getMessageId();
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
SQSBatchResponse.BatchItemFailure(messageId));
            }
        }
    }
}
```

```
    }
  }
  return new SQSBatchResponse(batchItemFailures);
}
}
```

## JavaScript

### SDK para JavaScript (v3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}
```

## Notificación de los errores de los elementos del lote de SQS con Lambda mediante TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
  from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
  Promise<SQSBatchResponse> => {
  const batchItemFailures: SQSBatchItemFailure[] = [];

  for (const record of event.Records) {
    try {
      await processMessageAsync(record);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }

  return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
  if (record.body && record.body.includes("error")) {
    throw new Error('There is an error in the SQS Message.');
```

## PHP

### SDK para PHP

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

## Notificación de los errores de los elementos del lote de SQS con Lambda mediante PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}
```

```
    }  
}  
  
$logger = new StderrLogger();  
return new Handler($logger);
```

## Python

### SDK para Python (Boto3)

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
def lambda_handler(event, context):  
    if event:  
        batch_item_failures = []  
        sqs_batch_response = {}  
  
        for record in event["Records"]:  
            try:  
                # process message  
            except Exception as e:  
                batch_item_failures.append({"itemIdentifier":  
record['messageId']})  
  
        sqs_batch_response["batchItemFailures"] = batch_item_failures  
        return sqs_batch_response
```



## Ruby

### SDK para Ruby

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
        rescue StandardError => e
          batch_item_failures << {"itemIdentifier" => record['messageId']}
        end
      end

      sqs_batch_response["batchItemFailures"] = batch_item_failures
      return sqs_batch_response
    end
  end
end
```

## Rust

### SDK para Rust

#### Note

Hay más información en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de [ejemplos sin servidor](#).

Notificación de los errores de los elementos del lote de SQS con Lambda mediante Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
    Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
run(service_fn(function_handler)).await  
}
```

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Ejemplos de servicios combinados de Lambda con SDK de AWS

Las siguientes aplicaciones de ejemplo utilizan SDK de AWS para combinar Lambda con otros Servicios de AWS. Cada ejemplo incluye un enlace a GitHub, con instrucciones de configuración y ejecución de la aplicación.

### Ejemplos

- [Creación de una API de REST de API Gateway para realizar un seguimiento de datos de COVID-19](#)
- [Creación de una API de REST de biblioteca de préstamos](#)
- [Creación de una aplicación de mensajería con Step Functions](#)
- [Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas](#)
- [Creación una aplicación de chat de websocket con API Gateway](#)
- [Creación de una aplicación que analice los comentarios de los clientes y sintetice el audio](#)
- [Invocación de una función Lambda desde un navegador](#)
- [Transformación de datos para su aplicación con S3 Object Lambda](#)
- [Uso de API Gateway para invocar una función de Lambda](#)
- [Uso de Step Functions para invocar funciones de Lambda](#)
- [Uso de eventos programados para invocar una función de Lambda](#)

## Creación de una API de REST de API Gateway para realizar un seguimiento de datos de COVID-19

En el siguiente ejemplo se muestra cómo crear una API REST que simule un sistema de seguimiento de los casos diarios de COVID-19 en Estados Unidos, con datos ficticios.

## Python

### SDK para Python (Boto3)

Muestra cómo utilizar AWS Chalice con AWS SDK for Python (Boto3) para crear una API REST sin servidor que utilice Amazon API Gateway, AWS Lambda y Amazon DynamoDB. La API REST simula un sistema que hace el seguimiento de los casos diarios de COVID-19 en Estados Unidos, con datos ficticios. Aprenda cómo:

- Utilizar AWS Chalice para definir rutas en las funciones de Lambda que se llaman para gestionar las solicitudes REST que llegan a través de API Gateway.
- Utilizar funciones de Lambda para recuperar y almacenar datos en una tabla de DynamoDB para atender solicitudes REST.
- Definir la estructura de tabla y los recursos del rol de seguridad en una plantilla de AWS CloudFormation.
- Utilizar AWS Chalice y CloudFormation para empaquetar e implementar todos los recursos necesarios.
- Utilizar CloudFormation para limpiar todos los recursos creados.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

#### Servicios utilizados en este ejemplo

- API Gateway
- AWS CloudFormation
- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Creación de una API de REST de biblioteca de préstamos

En el siguiente ejemplo de código se muestra cómo crear una biblioteca de préstamos en la que los usuarios puedan pedir prestados y devolver libros mediante una API de REST respaldada por una base de datos de Amazon Aurora.

## Python

### SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) con la API de Amazon Relational Database Service (Amazon RDS) y AWS Chalice para crear una API de REST respaldada por una base de datos de Amazon Aurora. El servicio web es totalmente sin servidor y representa una biblioteca de préstamos sencilla en la que los usuarios pueden pedir prestados libros y devolverlos. Aprenda cómo:

- Crear y administrar un clúster de base de datos Aurora sin servidor.
- Usar AWS Secrets Manager para administrar las credenciales de la base de datos.
- Implementar una capa de almacenamiento de datos que utilice Amazon RDS para mover datos dentro y fuera de la base de datos.
- Usar AWS Chalice para implementar una API de REST sin servidor en Amazon API Gateway y AWS Lambda.
- Utilice el paquete Requests para enviar solicitudes al servicio web.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

#### Servicios utilizados en este ejemplo

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Creación de una aplicación de mensajería con Step Functions

En el siguiente ejemplo se muestra cómo crear una aplicación de mensajería de AWS Step Functions que recupere registros de mensajes de una tabla de base de datos.

## Python

### SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) con AWS Step Functions para crear una aplicación de mensajería que recupere registros de mensajes de una tabla de Amazon DynamoDB y los envíe con Amazon Simple Queue Service (Amazon SQS). La máquina de estado se integra con una función de AWS Lambda para examinar la base de datos en busca de mensajes no enviados.

- Crear una máquina de estado que recupere y actualice los registros de mensajes de una tabla de Amazon DynamoDB.
- Actualizar la definición de la máquina de estado para que también envíe mensajes a Amazon Simple Queue Service (Amazon SQS).
- Iniciar y detener las ejecuciones de la máquina de estado.
- Conectar con Lambda, DynamoDB y Amazon SQS desde una máquina de estado mediante integraciones de servicio.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

#### Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas

En los siguientes ejemplos de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

## .NET

### AWS SDK for .NET

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## C++

### SDK para C++

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Java

### SDK para Java 2.x

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

### Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## JavaScript

### SDK para JavaScript (v3)

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).



## Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Kotlin

### SDK para Kotlin

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

## Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## PHP

### SDK para PHP

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

## Rust

### SDK para Rust

Muestra cómo desarrollar una aplicación de gestión de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Para profundizar en el origen de este ejemplo, consulte la publicación en [Comunidad de AWS](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Para obtener una lista completa de las guías para desarrolladores del SDK de AWS y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Creación una aplicación de chat de websocket con API Gateway

En el siguiente ejemplo se muestra cómo crear una aplicación de chat servida por una API de websocket basada en Amazon API Gateway.

### Python

#### SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) con Amazon API Gateway V2 para crear una API de websocket que se integre con AWS Lambda y Amazon DynamoDB.

- Crear una API de websocket servida por API Gateway.
- Definir un identificador Lambda que almacene las conexiones en DynamoDB y envíe mensajes a otros participantes del chat.
- Conectar con la aplicación de chat de websocket y enviar mensajes con el paquete Websockets.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

#### Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

# Creación de una aplicación que analice los comentarios de los clientes y sintetice el audio

En los siguientes ejemplos de código, se muestra cómo crear una aplicación que analice las tarjetas de comentarios de los clientes, las traduzca del idioma original, determine sus opiniones y genere un archivo de audio a partir del texto traducido.

## .NET

### AWS SDK for .NET

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en [GitHub](#).

### Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## Java

### SDK para Java 2.x

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel

recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en [GitHub](#).

Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## JavaScript

### SDK para JavaScript (v3)

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en [GitHub](#). Los siguientes extractos muestran cómo se usa AWS SDK for JavaScript dentro de las funciones de Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";
```

```

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
sourceDestinationConfig
 */

```

```
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});
```



```
const translateCommand = new TranslateTextCommand({
  SourceLanguageCode: textAndSourceLanguage.source_language_code,
  TargetLanguageCode: "en",
  Text: textAndSourceLanguage.extracted_text,
});

const { TranslatedText } = await translateClient.send(translateCommand);

return { translated_text: TranslatedText };
};
```

### Servicios utilizados en este ejemplo

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

## Ruby

### SDK para Ruby

Esta aplicación de ejemplo analiza y almacena las tarjetas de comentarios de los clientes. Concretamente, satisface la necesidad de un hotel ficticio en la ciudad de Nueva York. El hotel recibe comentarios de los huéspedes en varios idiomas en forma de tarjetas de comentarios físicas. Esos comentarios se cargan en la aplicación a través de un cliente web. Una vez cargada la imagen de una tarjeta de comentarios, se llevan a cabo los siguientes pasos:

- El texto se extrae de la imagen mediante Amazon Textract.
- Amazon Comprehend determina la opinión del texto extraído y su idioma.
- El texto extraído se traduce al inglés mediante Amazon Translate.
- Amazon Polly sintetiza un archivo de audio a partir del texto extraído.

La aplicación completa se puede implementar con AWS CDK. Para obtener el código fuente y las instrucciones de implementación, consulte el proyecto en [GitHub](#).

### Servicios utilizados en este ejemplo

- Amazon Comprehend

- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Para obtener una lista completa de las guías para desarrolladores del SDK de AWS y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Invocación de una función Lambda desde un navegador

En el siguiente ejemplo se muestra cómo invocar una función AWS Lambda desde un navegador.

### JavaScript

#### SDK para JavaScript (v2)

Puede crear una aplicación basada en el navegador que utilice una función AWS Lambda para actualizar una tabla de Amazon DynamoDB con las selecciones del usuario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

#### SDK para JavaScript (v3)

Puede crear una aplicación basada en el navegador que utilice una función AWS Lambda para actualizar una tabla de Amazon DynamoDB con las selecciones del usuario. Esta aplicación utiliza AWS SDK for JavaScript v3.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Transformación de datos para su aplicación con S3 Object Lambda

En el siguiente ejemplo de código se muestra cómo transformar datos para su aplicación con S3 Object Lambda.

.NET

### AWS SDK for .NET

Muestra cómo agregar código personalizado a las solicitudes GET S3 estándar para modificar el objeto solicitado recuperado de S3, de modo que el objeto se ajuste a las necesidades del cliente o aplicación solicitante.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- Lambda
- Amazon S3

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de API Gateway para invocar una función de Lambda

Los siguientes ejemplos de código muestran cómo crear una función AWS Lambda invocada por Amazon API Gateway.

Java

### SDK para Java 2.x

Indica cómo crear una función AWS Lambda utilizando la API de tiempo de ejecución de Java Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de

uso específico. En este ejemplo se indica cómo crear una función de Lambda invocada por Amazon API Gateway que escanea una tabla de Amazon DynamoDB en busca de aniversarios laborales y utiliza Amazon Simple Notification Service (Amazon SNS) para enviar un mensaje de texto a sus empleados que les felicite en la fecha de su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

## JavaScript

### SDK para JavaScript (v3)

Indica cómo crear una función de Lambda mediante el uso de la API de tiempo de ejecución de Lambda JavaScript. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. En este ejemplo se indica cómo crear una función de Lambda invocada por Amazon API Gateway que escanea una tabla de Amazon DynamoDB en busca de aniversarios laborales y utiliza Amazon Simple Notification Service (Amazon SNS) para enviar un mensaje de texto a sus empleados que les felicite en la fecha de su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

## Python

### SDK para Python (Boto3)

En este ejemplo se muestra cómo crear y utilizar una API de REST de Amazon API Gateway dirigida a una función AWS Lambda. El controlador Lambda muestra cómo enrutar según los métodos HTTP; cómo obtener datos de la cadena de consulta, el encabezado y el cuerpo; y cómo devolver una respuesta JSON.

- Implemente una función de Lambda.
- Cree una API de REST mediante API Gateway.
- Cree un recurso REST que se dirija a la función de Lambda.
- Otorgue permiso para permitir que API Gateway invoque la función de Lambda.
- Utilice el paquete Requests para enviar solicitudes a la API de REST.
- Limpie todos los recursos creados durante la demostración.

Este ejemplo se puede ver mejor en GitHub. Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

#### Servicios utilizados en este ejemplo

- API Gateway
- Lambda

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

## Uso de Step Functions para invocar funciones de Lambda

Los siguientes ejemplos de código muestran cómo crear una máquina de estado de AWS Step Functions que invoque funciones de AWS Lambda en secuencia.

### Java

#### SDK para Java 2.x

Muestra cómo crear un flujo de trabajo sin servidor de AWS con AWS Step Functions y AWS SDK for Java 2.x. Cada paso del flujo de trabajo se implementa con una función de AWS Lambda.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

## JavaScript

### SDK para JavaScript (v3)

Muestra cómo crear un flujo de trabajo sin servidor de AWS con AWS Step Functions y AWS SDK for JavaScript. Cada paso del flujo de trabajo se implementa con una función de Lambda.

Lambda es un servicio de computación que permite ejecutar código sin aprovisionar ni administrar servidores. Step Functions es un servicio de orquestación sin servidor que le permite combinar funciones de Lambda y otros servicios de AWS para crear aplicaciones esenciales desde el punto de vista empresarial.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

# Uso de eventos programados para invocar una función de Lambda

Los siguientes ejemplos de código muestran cómo crear una función AWS Lambda invocada por un evento programado de Amazon EventBridge.

## Java

### SDK para Java 2.x

Muestra cómo crear un evento programado de Amazon EventBridge que invoque una función de AWS Lambda. Configuración de EventBridge para que utilice una expresión cron para programar la invocación de la función de Lambda. En este ejemplo, creará una función de Lambda utilizando la API de tiempo de ejecución de Lambda Java. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Este ejemplo indica cómo crear una aplicación que envíe un mensaje de texto a sus empleados para felicitarles por su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

### Servicios utilizados en este ejemplo

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

## JavaScript

### SDK para JavaScript (v3)

Muestra cómo crear un evento programado de Amazon EventBridge que invoque una función de Lambda. Configuración de EventBridge para que utilice una expresión cron para programar la invocación de la función de Lambda. En este ejemplo, creará una función de Lambda utilizando la API de tiempo de ejecución de Lambda JavaScript. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Este ejemplo indica cómo crear una aplicación que envíe un mensaje de texto a sus empleados para felicitarles por su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Este ejemplo también está disponible en la [guía para desarrolladores de AWS SDK for JavaScript v3](#).

Servicios utilizados en este ejemplo

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

## Python

### SDK para Python (Boto3)

En este ejemplo, se muestra cómo registrar una función de AWS Lambda como destino de un evento de Amazon EventBridge programado. El controlador de Lambda escribe un mensaje sencillo y los datos de eventos completos en Registros de Amazon CloudWatch para recuperarlos posteriormente.

- Implementa una función de Lambda.
- Crea un evento programado de EventBridge y convierte la función de Lambda en el destino.
- Otorga permiso para permitir que EventBridge invoque la función de Lambda.
- Imprime los datos más recientes de Registros de CloudWatch para mostrar el resultado de las invocaciones programadas.
- Limpia todos los recursos creados durante la demostración.

Este ejemplo se puede ver mejor en GitHub. Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en [GitHub](#).

Servicios utilizados en este ejemplo

- Registros de CloudWatch
- EventBridge
- Lambda



Para obtener una lista completa de las guías para desarrolladores del AWS SDK y ejemplos de código, consulte [Utilización de Lambda con SDK de AWS](#). En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

# Cuotas de Lambda

## Important

Las nuevas Cuentas de AWS tienen cuotas de simultaneidad y memoria reducidas. AWS aumenta estas cuotas automáticamente en función del uso.

## Informática y almacenamiento

Lambda establece cuotas para la cantidad de recursos informáticos y de almacenamiento que puede usar para ejecutar y almacenar funciones. Las cuotas para las ejecuciones y el almacenamiento simultáneos se aplican por Región de AWS. Las cuotas de la interfaz de red elástica (ENI) se aplican por nube privada virtual (VPC), sin importar la región. Las siguientes cuotas se pueden aumentar con respecto a sus valores predeterminados. Para obtener más información, consulte [Solicitud de un aumento de cuota](#) en la Guía del usuario de Service Quotas.

Recurso	Cuota predeterminada	Se puede aumentar hasta
Ejecuciones simultáneas	1 000	Decenas de miles
Almacenamiento para funciones cargadas (archivos .zip) y capas. Cada versión de función y de capa consume almacenamiento.  Para conocer las prácticas recomendadas sobre la administración del almacenamiento de código, consulte <a href="#">Monitoring Lambda code storage</a> en Serverless Land.	75 GB	Terabytes
Almacenamiento para funciones definidas como imágenes de contenedores. Estas imágenes se almacenan en Amazon ECR.	Consulte <a href="#">Service Quotas de Amazon ECR</a> .	

Recurso	Cuota predeterminada	Se puede aumentar hasta
<a href="#">Interfaces de red elásticas por Virtual Private Cloud (VPC)</a> <div style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p><b>Note</b></p> <p>Esta cuota se comparte con otros servicios, como Amazon Elastic File System (Amazon EFS). Consulte <a href="#">Cuotas de Amazon VPC</a>.</p> </div>	250	Miles

Para obtener más información sobre la simultaneidad y cómo Lambda amplía la simultaneidad de funciones en respuesta al tráfico, consulte [Comprender el escalado de la función de Lambda](#).

## Configuración, implementación y ejecución de funciones

Las siguientes cuotas se aplican a la configuración, implementación y ejecución de funciones. Salvo que se indique lo contrario, no se pueden cambiar.

### Note

La documentación de Lambda, los mensajes de registro y la consola utilizan la abreviatura MB (en lugar de MiB) para hacer referencia a 1024 KB.


Recurso	Cuota
Asignación <a href="#">de memoria de función</a>	<p>De 128 MB a 10 240 MB, en incrementos de 1 MB.</p> <p>Nota: Lambda asigna potencia de CPU en proporción a la cantidad de memoria configurada. Para aumentar</p>

Recurso	Cuota
	o disminuir la memoria y la potencia de CPU asignada a su función, utilice la configuración Memoria (MB). En 1769 MB, la función tiene el equivalente de una vCPU.
Tiempo de espera de la función	900 segundos (15 minutos)
Función <a href="#">variables de entorno</a>	4 KB, para todas las variables de entorno asociadas a la función, en conjunto
<a href="#">Política basada en recursos</a> de la función	20 KB
Capas de <a href="#">funciones</a>	cinco capas
<a href="#">Límite de escalado de simultaneidad</a> de funciones	Para cada función, 1000 entornos de ejecución cada 10 segundos
<a href="#">Carga de invocación</a> (solicitud y respuesta)	<p>6 MB cada una para solicitud y respuesta (sincrónicas)</p> <p>20 MB para cada <a href="#">respuesta transmitida</a> (sincrónicas. El tamaño de la carga útil de las respuestas transmitidas se puede aumentar con respecto a los valores predeterminados. Póngase en contacto con AWS Support para obtener más información).</p> <p>256 KB (asíncronas)</p> <p>1 MB es el límite del tamaño total combinado de la línea de la solicitud y los valores del encabezado</p>

Recurso	Cuota
Ancho de banda para <a href="#">respuestas transmitidas</a>	Sin límite durante los primeros 6 MB de la respuesta de la función  Para respuestas de más de 6 MB, 2 Mbps para el resto de la respuesta
<a href="#">Tamaño del paquete de implementación (archivo de archivo .zip)</a>	50 MB (comprimido, para carga directa)  250 MB (descomprimido)  Esta cuota se aplica a todos los archivos que cargue, incluidas las capas y los tiempos de ejecución personalizados.  3 MB (editor de consola)
Tamaño de la configuración de imagen de contenedor	16 KB
Tamaño del paquete del código de <a href="#">imagen de contenedor</a>	10 GB (tamaño máximo de imagen sin comprimir, incluidas todas las capas)
Eventos de prueba (editor de consola)	10
/tmp almacenamiento del directorio	Entre 512 MB y 10 240 MB, en incrementos de 1 MB.
Descriptores de archivo	1 024
Ejecución procesos/subprocesos	1 024

## Solicitudes de la API de Lambda

Las cuotas siguientes están asociadas con solicitudes de la API de Lambda.

Recurso	Cuota
Solicitudes de invocación por función en cada región (sincrónicas)	Cada instancia de su entorno de ejecución puede atender hasta 10 solicitudes por segundo. En otras palabras, el límite total de invocaciones es 10 veces más alto que el límite de simultaneidad. Consulte <a href="#">Comprender el escalado de la función de Lambda</a> .
Solicitudes de invocación por función en cada región (asíncronas)	Cada instancia de su entorno de ejecución puede atender un número ilimitado de solicitudes. En otras palabras, el límite total de invocaciones se basa solo en la simultaneidad disponible para la función. Consulte <a href="#">Comprender el escalado de la función de Lambda</a> .
Solicitudes de invocación por versión de función o alias (solicitudes por segundo)	10 x <a href="#">simultaneidad aprovisionada</a> asignada  <div data-bbox="971 1205 1507 1474" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"><p> <b>Note</b></p><p>Esta cuota solo se aplica a funciones que utilizan la simultaneidad aprovisionada.</p></div>
Solicitudes de la API <a href="#">GetFunction</a>	100 solicitudes por segundo. No se puede aumentar.
Solicitudes de la API <a href="#">GetPolicy</a>	15 solicitudes por segundo. No se puede aumentar.

Recurso	Cuota
Resto de las solicitudes de la API del plano de control (excluye las solicitudes de invocación, GetFunction y GetPolicy)	15 solicitudes por segundo en todas las API (y no 15 solicitudes por segundo y por API). No se puede aumentar.

## Otros servicios

Cuotas para otros servicios, como AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda@Edge) y Amazon Virtual Private Cloud (Amazon VPC), pueden afectar sus funciones de Lambda. Para obtener más información, consulte [Cuotas de Servicio de AWS](#) en la Referencia general de Amazon Web Services y en [Invocar Lambda con eventos de otros servicios de AWS](#).

# Historial del documento

En la siguiente tabla, se describen los cambios importantes que se han realizado en la Guía para desarrolladores de AWS Lambda desde mayo de 2018. Para obtener notificaciones sobre las actualizaciones de esta documentación, suscríbase a la [fuente RSS](#).

Cambio	Descripción	Fecha
<a href="#">Compatibilidad con SnapStart en las nuevas regiones</a>	Lambda <a href="#">SnapStart</a> ahora está disponible en las siguientes regiones: Europa (España), Europa (Zúrich), Asia-Pacífico (Melbourne), Asia-Pacífico (Hyderabad) y Medio Oriente (Emiratos Árabes Unidos).	12 de enero de 2024
<a href="#">Actualizaciones de políticas administradas por AWS</a>	Service Quotas actualizó una política administrada por AWS existente (AWSLambdaVPCLambdaAccessExecutionRole ).	5 de enero de 2024
<a href="#">Tiempo de ejecución de Python 3.12</a>	Lambda ahora admite Python 3.12 como tiempo de ejecución administrado e imagen base de contenedor. Para obtener más información, consulte el tiempo de ejecución de <a href="#">Python 3.12, que ahora está disponible en AWS Lambda</a> en el Blog de informática de AWS.	14 de diciembre de 2023
<a href="#">Tiempo de ejecución en Java 21</a>	Lambda ahora admite Java 21 como tiempo de ejecución administrado e imagen base de contenedor (java21).	16 de noviembre de 2023



---

<a href="#">Tiempo de ejecución de Node.js 20</a>	Lambda ahora admite Java 20 como tiempo de ejecución administrado e imagen base de contenedor (nodejs20.x). Para obtener más información, consulte el tiempo de ejecución de <a href="#">Node.js 20.x, que ahora está disponible en AWS Lambda</a> en el Blog de informática de AWS.	14 de noviembre de 2023
<a href="#">Tiempo de ejecución provided al2023</a>	Lambda ahora admite Amazon Linux 2023 como tiempo de ejecución administrado e imagen base de contenedor. Para obtener más información, consulte <a href="#">Presentación del tiempo de ejecución de Amazon Linux 2023 para AWS Lambda</a> en el Blog de informática de AWS.	9 de noviembre de 2023
<a href="#">Compatibilidad de IPv6 para subredes de doble pila</a>	Lambda ahora admite tráfico IPv6 saliente a subredes de doble pila. Para obtener más información, consulte <a href="#">Compatibilidad de IPv6</a> .	12 de octubre de 2023

## [Prueba de funciones y aplicaciones sin servidor](#)

Aprenda las técnicas para depurar y automatizar las pruebas de funciones sin servidor en la nube. Ahora hay un capítulo de pruebas y recursos incluidos en las secciones de lenguaje Python y Typescript. Para obtener más información, consulte [Pruebas de funciones y aplicaciones sin servidor](#).

16 de junio de 2023

## [Tiempo de ejecución de Ruby 3.2](#)

Lambda ahora es compatible con un nuevo tiempo de ejecución de Ruby 3.2. Para obtener más información, consulte [Creación de funciones de Lambda con Ruby](#).

7 de junio de 2023

## [Transmisión de respuestas](#)

Lambda ahora es compatible con la transmisión de respuestas desde funciones . Para obtener más información, consulte [Configuración de una función de Lambda para transmitir respuestas](#).

6 de abril de 2023

## [Métricas de invocación asíncrona](#)

Lambda lanza métricas de invocación asíncrona. Para obtener más información, consulte [Métricas de invocación asíncrona](#).

9 de febrero de 2023

## [Controles de versión de tiempos de ejecución](#)

Lambda publica nuevas versiones de tiempos de ejecución que incluyen actualizaciones de seguridad, correcciones de errores y características nuevas. Ahora, puede controlar cuándo se actualizan sus funciones a las nuevas versiones de tiempo de ejecución. Para obtener más información, consulte [Lambda runtime updates](#) (Actualizaciones de tiempo de ejecución de Lambda).

23 de enero de 2023

## [Lambda SnapStart](#)

Utilice Lambda SnapStart para reducir el tiempo de inicio de las funciones Java sin aprovisionar recursos adicionales ni implementar optimizaciones de rendimiento complejas. Para obtener más información, consulte [Mejora del rendimiento de inicio con Lambda SnapStart](#).

28 de noviembre de 2022

## [Tiempo de ejecución de Node.js 18](#)

Lambda ahora es compatible con un nuevo tiempo de ejecución de Node.js 18. Node.js 18 utiliza Amazon Linux 2. Para obtener más información, consulte [Creación de funciones de Lambda con Node.js](#).

18 de noviembre de 2022

---

<a href="#">Clave de condición lambda:SourceFunctionArn</a>	Para un recurso de AWS, la clave de condición <code>lambda:SourceFunctionArn</code> filtra el acceso al recurso mediante el ARN de una función de Lambda. Para obtener más detalles, consulte <a href="#">Uso de las credenciales del entorno de ejecución de Lambda</a> .	1 de julio de 2022
<a href="#">Tiempo de ejecución de Node.js 16</a>	Lambda ahora es compatible con un nuevo tiempo de ejecución de Node.js 16. Node.js 16 utiliza Amazon Linux 2. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Node.js</a> .	11 de mayo de 2022
<a href="#">URL de funciones de Lambda</a>	Lambda ahora es compatible con las URL de funciones, que son puntos de conexión HTTP(S) dedicados para las funciones de Lambda. Para obtener más información, consulte <a href="#">URL de funciones de Lambda</a> .	6 de abril de 2022
<a href="#">Eventos de prueba compartidos en la consola de AWS Lambda</a>	Lambda ahora es compatible con el uso compartido de eventos de prueba con otros usuarios en la misma Cuenta de AWS. Para obtener más información, consulte <a href="#">Probar la función de Lambda con la consola</a> .	16 de marzo de 2022

[PrincipalOrgId en políticas basadas en recursos](#)

Lambda ahora es compatible con la concesión de permisos a una organización en AWS Organizations. Para obtener más información, consulte [Uso de políticas basadas en recursos para AWS Lambda](#).

11 de marzo de 2022

[Tiempo de ejecución de .NET 6](#)

Lambda ahora es compatible con un nuevo tiempo de ejecución de .NET 6. Para obtener más información, consulte [Tiempos de ejecución de Lambda](#).

23 de febrero de 2022

[Filtrado de eventos de los orígenes de eventos de Kinesis, DynamoDB y Amazon SQS](#)

Lambda ahora es compatible con el filtrado de eventos de los orígenes de eventos de Kinesis, DynamoDB y Amazon SQS. Para obtener más información, consulte [Filtrado de eventos de Lambda](#).

24 de noviembre de 2021

[Autenticación de mTLS para orígenes de eventos de Amazon MSK y autoadministrados de Apache Kafka](#)

Lambda ya admite la autenticación de mTLS para orígenes de eventos de Amazon MSK y autoadministrados de Apache Kafka. Para obtener más información, consulte [Uso de Lambda con Amazon MSK](#).

19 de noviembre de 2021

---

<a href="#">Lambda en Graviton2</a>	Lambda ahora es compatible con Graviton2 para funciones que utilizan arquitectura arm64. Para obtener más información, consulte <a href="#">Arquitecturas del conjunto de instrucciones Lambda</a> .	29 de septiembre de 2021
<a href="#">Tiempo de ejecución de Python 3.9</a>	Lambda ahora es compatible con un nuevo tiempo de ejecución de Python 3.9. Para obtener más información, consulte <a href="#">Tiempos de ejecución de Lambda</a> .	16 de agosto de 2021
<a href="#">Nuevas versiones de tiempos de ejecución para Node.js, Python y Java</a>	Hay nuevas versiones de tiempos de ejecución disponibles para Node.js, Python y Java. Para obtener más información, consulte <a href="#">Tiempos de ejecución de Lambda</a> .	21 de julio de 2021
<a href="#">Soporte para RabbitMQ como origen de eventos en Lambda</a>	Lambda ahora admite Amazon MQ para RabbitMQ como fuente de eventos. Amazon MQ es un servicio de agente de mensajes administrado en Apache ActiveMQ y RabbitMQ que facilita la configuración y el funcionamiento de los agentes de mensajes en la nube. Para obtener más información, consulte <a href="#">Uso de Lambda con Amazon MQ</a> .	7 de julio de 2021

### [Autenticación SASL/PLAIN para Kafka autoadministrado en Lambda](#)

SASL/PLAIN es ahora un mecanismo de autenticación compatible para fuentes de eventos Kafka autoadministradas en Lambda Customers que ya utilizan SASL/PLAIN en su clúster de Kafka autoadministrado ahora pueden utilizar fácilmente Lambda para crear aplicaciones de consumo sin tener que modificar la forma en que se autentican. Para obtener más información, consulte [Uso de Lambda con Apache Kafka autoadministrado](#).

29 de junio de 2021

### [API de extensiones de Lambda](#)

Disponibilidad general para extensiones de Lambda. Use las extensiones para aumentar las funciones Lambda. Puede usar las extensiones proporcionadas por socios de Lambda o puede crear sus propias extensiones de Lambda. Para obtener más información, consulte [API de extensiones de Lambda](#).

24 de mayo de 2021

### [Nueva experiencia de consola de Lambda](#)

La consola Lambda se ha rediseñado para mejorar el rendimiento y la consistencia.

2 de marzo de 2021

## [Tiempo de ejecución de Node.js 14](#)

Lambda ahora es compatible con un nuevo tiempo de ejecución de Node.js 14. Node.js 14 utiliza Amazon Linux 2. Para obtener más información, consulte [Creación de funciones de Lambda con Node.js](#).

27 de enero de 2021

## [Imágenes de contenedor Lambda](#)

Lambda ahora soporta funciones definidas como imágenes de contenedor. Puede combinar la flexibilidad de las herramientas de contenedores con la agilidad y simplicidad operativa de Lambda crear aplicaciones. Para obtener más información, consulte [Uso de imágenes de contenedor con Lambda](#).

1 de diciembre de 2020



## [Firma de código para funciones de Lambda](#)

Lambda ahora es compatible con la firma de código. Los administradores pueden configurar funciones de Lambda para que acepten solo código firmado en la implementación. Lambda comprueba las firmas para asegurarse de que el código no se altera o modifica. Además, Lambda se asegura de que el código esté firmado por desarrolladores de confianza antes de aceptar la implementación. Para obtener más información, consulte [Configuración de la firma de código para Lambda](#).

23 de noviembre de 2020

## [Vista previa: API de registros de tiempo de ejecución de Lambda](#)

Lambda ahora admite la API de registros de tiempo de ejecución. Las extensiones de Lambda se pueden usar con la API de registros para suscribirse a flujos de registro en el entorno de ejecución. Para obtener más información, consulte la [API de registros de tiempo de ejecución de Lambda](#).

12 de noviembre de 2020

---

<a href="#">Nuevo origen de eventos para Amazon MQ</a>	Lambda ahora admite Amazon MQ como fuente de eventos. Utilice una función de Lambda para procesar registros de su agente de mensajes Amazon MQ. Para obtener más información, consulte <a href="#">Uso de Lambda con Amazon MQ</a> .	5 de noviembre de 2020
<a href="#">Vista previa: API de extensiones de Lambda</a>	Puede usar extensiones de Lambda para aumentar las funciones Lambda. Puede usar las extensiones proporcionadas por socios de Lambda o puede crear sus propias extensiones de Lambda. Para obtener más información, consulte <a href="#">API de extensiones de Lambda</a> .	8 de octubre de 2020
<a href="#">Compatibilidad con Java 8 y tiempos de ejecución personalizados en AL2</a>	Lambda ahora admite Java 8 y tiempos de ejecución personalizados en Amazon Linux 2. Para obtener más información, consulte <a href="#">Tiempos de ejecución de Lambda</a> .	12 de agosto de 2020
<a href="#">Nuevo origen de eventos para Amazon Managed Streaming for Apache Kafka</a>	Lambda ahora admite Amazon MSK como fuente de eventos. Utilice una función de Lambda con Amazon MSK para procesar registros en un tema de Kafka. Para obtener más información, consulte <a href="#">Uso de Lambda con Amazon MSK</a> .	11 de agosto de 2020

[Claves de condición de IAM para la configuración de Amazon VPC](#)

Ahora puede usar claves de condición específicas de Lambda para la configuración de VPC. Por ejemplo, puede requerir que todas las funciones de la organización estén conectadas a una VPC. También puede especificar las subredes y los grupos de seguridad que los usuarios de la función pueden y no pueden utilizar. Para obtener más información, consulte [Configuración de VPC para funciones de IAM](#).

10 de agosto de 2020

[Configuración de concurrencia para consumidores de transmisiones HTTP/2 de Kinesis](#)

Ahora puede usar la siguiente configuración de concurrencia para consumidores de Kinesis con distribución ramificada a mejorada (secuencias HTTP/2): `ParallelizationFactor`, `MaximumRetryAttempts`, `MaximumRecordAgeInSeconds`, `DestinationConfig` y `BisectBatchOnFunctionError`. Para obtener más información, consulte [Uso de AWS Lambda con Amazon Kinesis](#).

7 de julio de 2020

[Ventana por lotes para consumidores de transmisiones HTTP/2 de Kinesis](#)

Ahora puede configurar una ventana por lotes (MaximumBatchingWindowInSeconds) para flujos HTTP/2. Lambda lee los registros del flujo hasta que haya recopilado un lote completo o hasta que caduque la ventana del lote. Para obtener más información, consulte [Uso de AWS Lambda con Amazon Kinesis](#).

18 de junio de 2020

[Compatibilidad con sistemas de archivos de Amazon EFS](#)

Ahora puede conectar un sistema de archivos de Amazon EFS a sus funciones de Lambda para acceder a archivos de red compartidos. Para obtener más información, consulte [Configuración del acceso al sistema de archivos para las funciones de Lambda](#).

16 de junio de 2020

[Aplicaciones de ejemplo de AWS CDK en la consola de Lambda](#)

La consola de Lambda ahora incluye aplicaciones de ejemplo que utilizan AWS Cloud Development Kit (AWS CDK) para TypeScript. El AWS CDK es un marco que le permite definir los recursos de su aplicación en TypeScript, Python, Java o .NET.

1 de junio de 2020

[Compatibilidad con el tiempo de ejecución de .NET Core 3.1.0 en AWS Lambda](#)

AWS Lambda ahora admite el tiempo de ejecución de .NET Core 3.1.0. Para obtener más información, consulte [CLI de .NET Core](#).

31 de marzo de 2020

## [Compatibilidad con las API HTTP de API Gateway](#)

Se ha actualizado y ampliado la documentación sobre el uso de Lambda con API Gateway, como la compatibilidad con las API HTTP. Se ha incorporado una aplicación de ejemplo que crea una API y una función con AWS CloudFormation. Para obtener más información, consulte [Uso de Lambda con Amazon API Gateway](#).

23 de marzo de 2020

## [Ruby 2.7](#)

Hay un nuevo tiempo de ejecución disponible para Ruby 2.7, ruby2.7, que es el primer tiempo de ejecución de Ruby que utiliza Amazon Linux 2. Para obtener más información, consulte [Creación de funciones Lambda con Ruby](#).

19 de febrero de 2020

## Métricas de simultaneidad

Ahora Lambda informa de la métrica de `ConcurrentExecutions` para todas las funciones, alias y versiones. Puede ver un gráfico para esta métrica en la página de supervisión de su función. Anteriormente, solo se informó de `ConcurrentExecutions` en el nivel de cuenta y para las funciones que utilizan concurrencia reservada. Para obtener más detalles, consulte [Métricas de función de AWS Lambda](#).

18 de febrero de 2020

## [Actualizar a estados de la función](#)

24 de enero de 2020

Los estados de función se aplican ahora para todas las funciones de forma predeterminada. Al conectar una función a una VPC, Lambda crea interfaces de redes elásticas compartidas. Esto permite que su función se escale sin crear interfaces de red adicionales. Durante este tiempo, no puede realizar operaciones adicionales en la función, incluida la actualización de su configuración y la publicación de versiones. En algunos casos, la invocación también se ve afectada. Los detalles sobre el estado actual de una función están disponibles en la API de Lambda.

Esta actualización se está publicando en fases. Para obtener más información, consulte el [Ciclo de vida actualizado de los estados de Lambda para redes de VPC](#) en el blog de informática de AWS. Para obtener más información acerca de los estados, consulte [Estados de función de AWS Lambda](#).

### [Actualizaciones a la salida de la API de configuración de funciones](#)

Se han agregado códigos de motivo a [StateReasonCode](#) (InvalidSubnet, InvalidSecurityGroup) y LastUpdateStatusReasonCode (SubnetOutOfIPAddresses, InvalidSubnet, InvalidSecurityGroup) para las funciones que se conectan a una VPC. Para obtener más información acerca de los estados, consulte [Estados de función de AWS Lambda](#).

20 de enero de 2020

### [Simultaneidad aprovisionada](#)

Ahora puede asignar simultaneidad aprovisionada para una versión o alias de función. La simultaneidad aprovisionada permite que una función escale sin fluctuaciones en la latencia. Para obtener más detalles, consulte [Administración de la simultaneidad de una función de Lambda](#).

3 de diciembre de 2019



### [Crear un proxy de base de datos](#)

Ahora puede usar la consola de Lambda para crear un proxy de base de datos para una función Lambda. Un proxy de base de datos permite que una función alcance altos niveles de simultaneidad sin agotar las conexiones de base de datos. Para obtener más información, consulte [Configuración del acceso a la base de datos para una función Lambda](#).

3 de diciembre de 2019

### [Compatibilidad con percentiles para la métrica de duración](#)

Ahora puede filtrar la métrica de duración en función de los percentiles. Para obtener más información, consulte [Métricas de AWS Lambda](#).

26 de noviembre de 2019

### [Aumento de la simultaneidad para orígenes de eventos de transmisión](#)

Una nueva opción para mapeos de orígenes de eventos de [flujos de DynamoDB](#) y de [flujos de Kinesis](#) le permiten procesar más de un lote a la vez de cada partición. Cuando aumenta el número de lotes simultáneos por partición, la simultaneidad de su función puede ser hasta diez veces mayor que el número de particiones en el flujo. Para obtener más detalles, consulte [Mapeo de fuentes de eventos de Lambda](#).

25 de noviembre de 2019

## [Estados de la función](#)

Al crear o actualizar una función, esta cambia a un estado pendiente mientras Lambda aprovisiona recursos para admitirla. Si conecta su función a una VPC, Lambda puede crear una interfaz de red elástica compartida de inmediato, en lugar de crear interfaces de red cuando se invoca la función. Esto da como resultado un mejor rendimiento para las funciones conectadas a la VPC, pero podría requerir una actualización de la automatización. Para obtener más detalles, consulte [Estados de función de AWS Lambda](#).

25 de noviembre de 2019

## [Opciones de gestión de errores para la invocación asincrónica](#)

Hay disponibles nuevas opciones de configuración para la invocación asíncrona. Puede configurar Lambda para limitar los reintentos y establecer una antigüedad de evento máxima. Para obtener más información, consulte [Configuración de la gestión de errores para la invocación asíncrona](#).

25 de noviembre de 2019

## [Gestión de errores para orígenes de eventos de transmisión](#)

Hay disponibles nuevas opciones de configuración para los mapeos de origen de eventos que se leen de los flujos. Puede configurar los mapeos de fuentes de eventos de [flujos de DynamoDB](#) y de [flujos de Kinesis](#) para limitar los reintentos y establecer una antigüedad de registro máxima. Cuando se producen errores, puede configurar la asignación de orígenes de eventos para que divida los lotes antes de volver a intentarlo y envíe registros de invocación de los lotes con error a una cola o tema. Para obtener más detalles, consulte [Mapeo de fuentes de eventos de Lambda](#).

25 de noviembre de 2019

## [Destinos para invocación asincrónica](#)

Ahora puede configurar Lambda para que envíe registros de invocaciones asincrónicas a otro servicio. Los registros de invocación contienen detalles sobre el evento, el contexto y la respuesta de la función. Puede enviar registros de invocación a una cola SQS, un tema SNS, una función de Lambda o un bus de eventos de EventBridge. Para obtener más información, consulte [Configuración de destinos para invocación asíncrona](#).

25 de noviembre de 2019

## [Nuevos tiempos de ejecución para Node.js, Python y Java](#)

Hay nuevos tiempos de ejecución disponibles para Node.js 12, Python 3.8 y Java 11. Para obtener más información, consulte [Tiempos de ejecución de Lambda](#).

18 de noviembre de 2019

## [Compatibilidad de la cola FIFO con los orígenes de eventos de Amazon SQS](#)

Ahora puede crear una asignación de orígenes de eventos que haga lecturas desde una cola de primero en entrar, primero en salir (FIFO). Antes, solo se admitían colas estándar. Para obtener más información, consulte [Uso de Lambda con Amazon SQS](#).

18 de noviembre de 2019

[Creación de aplicaciones en la consola de Lambda](#)

La creación de aplicaciones en la consola de Lambda ya está disponible en general. Para obtener instrucciones, consulte [Administrar aplicaciones en la consola Lambda](#).

31 de octubre de 2019

[Creación de aplicaciones en la consola de Lambda \(beta\)](#)

Ahora puede crear una aplicación de Lambda con una canalización de entrega continua integrada en la consola de Lambda. La consola proporciona aplicaciones de ejemplo que puede utilizar como punto de partida para su propio proyecto. Elija entre AWS CodeCommit y GitHub para el control de origen. Cada vez que se introducen cambios en el repositorio, la canalización incluida se genera e implementa automáticamente. Para obtener instrucciones, consulte [Administrar aplicaciones en la consola Lambda](#).

3 de octubre de 2019

## [Mejoras de rendimiento para funciones conectadas a una VPC](#)

3 de septiembre de 2019

Lambda utiliza ahora un nuevo tipo de interfaz de red elástica que comparten todas las funciones en una subred de virtual private cloud (VPC). Al conectar una función a una VPC, Lambda crea una interfaz de red para cada combinación de grupo de seguridad y subred que elija. Cuando las interfaces de red compartidas están disponibles, la función ya no tiene que crear interfaces de red adicionales conforme se escala verticalmente. Esto mejora drásticamente los tiempos de inicio. Para obtener más información, consulte [Configuración de una función de Lambda para obtener acceso a los recursos en una VPC](#).

### [Configuración de lotes de transmisión](#)

Ahora puede configurar una ventana de lote para los mapeos de fuentes de eventos de [Amazon DynamoDB](#) y [Amazon Kinesis](#). Puede configurar una ventana de lote de hasta cinco minutos para almacenar los registros entrantes hasta completar un lote. Esto reduce el número de invocaciones a la función cuando el flujo se encuentra menos activo.

29 de agosto de 2019

### [Integración de Información de registros de CloudWatch](#)

La página de monitorización de la consola de Lambda ahora incluye informes de Información de registros de Amazon CloudWatch. Para obtener más información, consulte [Funciones de monitorización en la consola de AWS Lambda](#).

18 de junio de 2019

### [Amazon Linux 2018.03](#)

El entorno de ejecución de Lambda se está actualizando para utilizar Amazon Linux 2018.03. Para obtener más información, consulte [Entorno de ejecución](#).

21 de mayo de 2019

## [Node.js 10](#)

Hay un nuevo tiempo de ejecución disponible para Node.js 10, nodejs10.x. Este tiempo de ejecución utiliza Node.js 10.15 y se actualizará con la última versión secundaria de Node.js 10 periódicamente. Node.js 10 es también el primer tiempo de ejecución que utiliza Amazon Linux 2. Para obtener más información, consulte [Creación de funciones de Lambda con Node.js](#).

13 de mayo de 2019

## [GetLayerVersionByArn API](#)

Utilice la API [GetLayerVersionByArn](#) para descargar información de la versión de capa con el ARN de versión como entrada. En comparación con [GetLayerVersion](#), [GetLayerVersionByArn](#) le permite utilizar el ARN directamente en lugar de analizarlo para obtener el nombre de la capa y el número de versión.

25 de abril de 2019

## [Ruby](#)

AWS Lambda ahora admite Ruby 2.5 con un nuevo tiempo de ejecución. Para obtener más información, consulte [Creación de funciones Lambda con Ruby](#).

29 de noviembre de 2018



[Capas](#)

Con las capas de Lambda, puede empaquetar e implementar bibliotecas, tiempos de ejecución personalizados y otras dependencias de forma independiente del código de la función. Comparta sus capas con sus otras cuentas o con todo el mundo. Para obtener más información, consulte [Capas de Lambda](#).

29 de noviembre de 2018

[Tiempos de ejecución personalizados](#)

Cree un tiempo de ejecución personalizado para ejecutar funciones de Lambda en su lenguaje de programación preferido. Para obtener más información, consulte [Tiempos de ejecución de Lambda personalizados](#).

29 de noviembre de 2018

[Desencadenadores de Application Load Balancer](#)

Elastic Load Balancing ahora es compatible con las funciones de Lambda como destino para los balanceadores de carga de aplicaciones. Para obtener más información, consulte [Uso de Lambda con balanceadores de carga de aplicaciones](#).

29 de noviembre de 2018

[Uso de los consumidores de transmisiones HTTP/2 de Kinesis como desencadenadores](#)

Puede utilizar los consumidores de flujos de datos HTTP/2 de Kinesis para enviar eventos a AWS Lambda. Los consumidores de flujos tienen rendimiento de lectura dedicado desde cada partición del flujo de datos y utilizan HTTP/2 para minimizar la latencia. Para obtener más información, consulte [Uso de Lambda con Kinesis](#).

19 de noviembre de 2018

[Python 3.7](#)

AWS Lambda ahora es compatible con Python 3.7 con un nuevo tiempo de ejecución. Para obtener más información, consulte [Creación de funciones Lambda con Python](#).

19 de noviembre de 2018

[Incremento del límite de carga para las invocaciones asíncronas de funciones](#)

El tamaño de carga máximo para las invocaciones asíncronas ha aumentado de 128 KB a 256 KB, que coincide con el tamaño máximo del mensaje desde un desencadenador de Amazon SNS. Para obtener detalles, consulte [Cuotas de Lambda](#).

16 de noviembre de 2018

[Región GovCloud \(EE. UU. Este\) de AWS](#)

AWS Lambda ya está disponible en la región AWS GovCloud (Este de EE. UU.).

12 de noviembre de 2018

[Temas de AWS SAM trasladados a la nueva Guía para desarrolladores](#)

Algunos contenidos se centraban en la creación de aplicaciones sin servidor usando el AWS Serverless Application Model (AWS SAM). Estos temas se han trasladado a la [Guía para desarrolladores de AWS Serverless Application Model](#).

25 de octubre de 2018

[Visualización de aplicaciones de Lambda en la consola](#)

Puede ver el estado de sus aplicaciones de Lambda en la página [Applications \(Aplicaciones\)](#) en la consola de Lambda. Esta página muestra el estado de la pila de AWS CloudFormation. Incluye enlaces a páginas donde puede ver más información acerca de los recursos de la pila. También puede ver métricas acumuladas para la aplicación y crear paneles de monitorización personalizados.

11 de octubre de 2018

[Límite de tiempo de espera de ejecución de funciones](#)

Para permitir las funciones de ejecución prolongada, el máximo tiempo de espera de ejecución configurable se ha aumentado de 5 minutos a 15 minutos. Para obtener más información, consulte [Límites de Lambda](#).

10 de octubre de 2018

<a href="#">Compatibilidad con lenguaje de PowerShell Core en AWS Lambda</a>	AWS Lambda ahora admite el lenguaje de PowerShell Core. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con PowerShell</a> .	11 de septiembre de 2018
<a href="#">Compatibilidad con el tiempo de ejecución de .NET Core 2.1.0 en AWS Lambda</a>	AWS Lambda ahora admite el tiempo de ejecución de .NET Core 2.1.0. Para obtener más información, consulte <a href="#">CLI de .NET Core</a> .	9 de julio de 2018
<a href="#">Actualizaciones ahora disponibles sobre RSS</a>	Ahora puede suscribirse a una fuente RSS para seguir las versiones de esta guía.	5 de julio de 2018
<a href="#">Compatibilidad con Amazon SQS como origen de eventos</a>	AWS Lambda ahora admite Amazon Simple Queue Service (Amazon SQS) como fuente de eventos. Para obtener más información, consulte <a href="#">Invocación de funciones de Lambda</a> .	28 de junio de 2018
<a href="#">Región China (Ningxia)</a>	AWS Lambda está ahora disponible en la región China (Ningxia) Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	28 de junio de 2018

## Actualizaciones anteriores

En la siguiente tabla, se describen los cambios importantes que se han realizado en cada una de las versiones de la Guía para desarrolladores de AWS Lambda anteriores a junio de 2018.

Cambio	Descripción	Fecha
Compatibilidad con tiempo de ejecución 8.10 de Node.js	AWS Lambda ahora es compatible con la versión 8.10 de tiempo de ejecución de Node.js. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Node.js</a> .	2 de abril de 2018
ID de revisión de los alias y las funciones	AWS Lambda admite ahora los ID de revisión en los alias y las versiones de las funciones. Puede usar estos ID para realizar un seguimiento de las actualizaciones condicionales y aplicarlas al actualizar los recursos de alias o la versión de función.	25 de enero de 2018
Soporte del tiempo de ejecución para Go y .NET 2.0	AWS Lambda ha agregado compatibilidad del tiempo de ejecución para Go y .NET 2.0. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Go</a> y <a href="#">Creación de funciones Lambda con C#</a> .	15 de enero de 2018
Rediseño de la consola	AWS Lambda presenta una nueva consola de Lambda para simplificar su experiencia y se ha añadido un Cloud9 Code Editor para mejorar su capacidad de depurar y revisar el código de las funciones.	30 de noviembre de 2017
Definición de límites de simultaneidad para cada función	AWS Lambda admite ahora la definición de límites de simultaneidad para cada función. Para obtener más información, consulte <a href="#">Configurar la simultaneidad reservada para una función</a> .	30 de noviembre de 2017
Desvío de tráfico mediante alias	AWS Lambda admite ahora el desvío de tráfico con alias. Para obtener más información, consulte <a href="#">Creación de implementaciones continuas para las funciones de Lambda</a> .	28 de noviembre de 2017

Cambio	Descripción	Fecha
Implementación de código gradual	AWS Lambda permite ahora implementar de forma segura nuevas versiones de su función de Lambda utilizando Code Deploy. Para obtener más información, consulte <a href="#">Implementación de código gradual</a> .	28 de noviembre de 2017
Región China (Pekín)	AWS Lambda está ahora disponible en la región China (Pekín) Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	9 de noviembre de 2017
Presentación de SAM Local	AWS Lambda presenta SAM Local (denominado también SAM CLI), una herramienta de la AWS CLI que proporciona un entorno para desarrollar, probar y analizar localmente aplicaciones sin servidor antes de cargarlas en el motor de tiempo de ejecución de Lambda. Para obtener más información, consulte el tema relacionado con la <a href="#">prueba y depuración de aplicaciones sin servidor</a> .	11 de agosto de 2017
Región de Canadá (centro)	AWS Lambda está disponible ahora en la región de Canadá (centro). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	22 de junio de 2017
Región de América del Sur (São Paulo)	AWS Lambda ahora está disponible en la región América del Sur (São Paulo). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	6 de junio de 2017
Compatibilidad de AWS Lambda para AWS X-Ray.	Lambda introduce la compatibilidad con X-Ray, que permite detectar, analizar y optimizar problemas de desempeño con las aplicaciones de Lambda. Para obtener más información, consulte <a href="#">Visualice las invocaciones de la función de Lambda mediante AWS X-Ray</a> .	19 de abril de 2017

Cambio	Descripción	Fecha
Región de Asia-Pacífico (Bombay)	AWS Lambda ahora está disponible en la región Asia Pacífico (Bombay). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	28 de marzo de 2017
AWS Lambda ahora es compatible con el tiempo de ejecución v6.10 de Node.js	AWS Lambda ha agregado compatibilidad con el tiempo de ejecución v6.10 de Node.js. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Node.js</a> .	22 de marzo de 2017
Región de Europa (Londres)	AWS Lambda ya está disponible en la región Europa (Londres). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	1 de febrero de 2017
Compatibilidad de AWS Lambda con el tiempo de ejecución de .NET, Lambda@Edge (vista previa), las colas de mensajes fallidos y la implementación automatizada de las aplicaciones sin servidor	<p>AWS Lambda incorpora compatibilidad con C#. Para obtener más información, consulte <a href="#">Creación de funciones Lambda con C#</a>.</p> <p>Lambda@Edge le permite ejecutar funciones de en las ubicaciones de borde de AWS en respuesta a eventos de CloudFront. Para obtener más información, consulte <a href="#">Uso AWS Lambda con CloudFront Lambda @Edge</a>.</p>	3 de diciembre de 2016
AWS Lambda agrega Amazon Lex como fuente de eventos compatible.	Con Lambda y Amazon Lex, puede crear rápidamente bots de chat para diversos servicios, como Slack y Facebook.	30 de noviembre de 2016

Cambio	Descripción	Fecha
Región del oeste de EE. UU. (Norte de California)	AWS Lambda ya está disponible en la región Oeste de EE. UU. (Norte de California). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	21 de noviembre de 2016
Se presenta AWS SAM, que permite crear e implementar aplicaciones basadas en Lambda y utilizar variables de entorno para los ajustes de configuración de las funciones de Lambda.	<p>AWS SAM: ahora puede utilizar AWS SAM para definir la sintaxis para expresar recursos dentro de una aplicación sin servidor. Para implementar la aplicación, solo tiene que especificar los recursos que necesita que formen parte de ella, junto con sus políticas de permisos asociadas en un archivo de plantilla de AWS CloudFormation (escrito en JSON o YAML), empaquetar sus artefactos de implementación e implementar la plantilla.</p> <p>Variables de entorno: puede utilizar variables de entorno para especificar las opciones de configuración de una función de Lambda fuera del código de la función. Para obtener más información, consulte <a href="#">Utilice variables de entorno Lambda para configurar valores en el código</a>.</p>	18 de noviembre de 2016
Región de Asia-Pacífico (Seúl)	AWS Lambda ya está disponible en la región Asia-Pacífico (Seúl). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	29 de agosto de 2016
Región de Asia-Pacífico (Sídney)	Lambda ya está disponible en la región Asia-Pacífico (Sídney). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	23 de junio de 2016
Actualizaciones de la consola de Lambda	La consola de Lambda se ha actualizado para simplificar el proceso de creación de roles.	23 de junio de 2016



Cambio	Descripción	Fecha
AWS Lambda ahora es compatible con el tiempo de ejecución v4.3 de Node.js	AWS Lambda ha agregado compatibilidad con el tiempo de ejecución v4.3. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Node.js</a> .	07 de abril de 2016
Región de Europa (Fráncfort)	Lambda ya está disponible en la región de Europa (Fráncfort). Para obtener más información acerca de las regiones y los puntos de conexión de Lambda, consulte <a href="#">Regiones y puntos de conexión</a> en Referencia general de AWS.	14 de marzo de 2016
Compatibilidad con VPC	Ahora puede configurar una función de Lambda para obtener acceso a los recursos de una VPC. Para obtener más información, consulte <a href="#">Otorgamiento a las funciones de Lambda de acceso a los recursos de una Amazon VPC</a> .	11 de febrero de 2016
Se ha actualizado el tiempo de ejecución de Lambda.	El <a href="#">entorno de ejecución</a> se ha actualizado.	4 de noviembre de 2015

Cambio	Descripción	Fecha
Compatibilidad con el control de versiones, Python para desarrollar código para las funciones de Lambda, eventos programados y aumento del tiempo de ejecución	<p>Ahora puede desarrollar el código de las funciones de Lambda con Python. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Python</a>.</p> <p>Control de versiones: puede mantener una o varias versiones de una función de Lambda. El control de versiones permite controlar qué versión de una función de Lambda se ejecuta en los distintos entornos (por ejemplo, desarrollo, pruebas, o producción). Para obtener más información, consulte <a href="#">Administrar las versiones de la función de Lambda</a>.</p> <p>Eventos programados: también puede configurar Lambda para que invoque el código de forma periódica y programada utilizando la consola de Lambda. Puede especificar una frecuencia fija (un número de horas, días o semanas) o una expresión cron. Para obtener más información, consulte <a href="#">Invocación de una función de Lambda según una programación</a>.</p> <p>Aumento del tiempo de ejecución: a partir de ahora, puede configurar las funciones de Lambda para que se ejecuten durante un máximo de cinco minutos, lo que permite utilizar funciones con tiempos de ejecución más largos como, por ejemplo, trabajos de adquisición y procesamiento de grandes volúmenes de datos.</p>	08 de octubre de 2015

Cambio	Descripción	Fecha
Compatibilidad con DynamoDB Streams	DynamoDB Streams ya está disponible con carácter general y se puede utilizar en todas las regiones en las que está disponible. Puede activar DynamoDB Streams para una tabla y utilizar una función de Lambda como un disparador para la tabla. Los disparadores son acciones personalizadas que se llevan a cabo en respuesta a las actualizaciones realizadas en la tabla de DynamoDB. Para ver un tutorial de ejemplo, consulte <a href="#">Tutorial: Uso de AWS Lambda con Amazon DynamoDB Streams</a> .	14 de julio de 2015
Lambda ahora permite invocar funciones de Lambda con clientes compatibles con REST.	<p>Hasta ahora, para invocar una función de Lambda desde aplicaciones web, móviles o IoT, se necesitaban los SDK de AWS (por ejemplo, AWS SDK for Java, AWS SDK for Android o AWS SDK for iOS). Ahora, Lambda permite invocar una función de Lambda con clientes compatibles con REST a través de una API personalizada que puede crear con Amazon API Gateway. Puede enviar solicitudes a la URL del punto de conexión de una función de Lambda. Puede configurar la seguridad en el punto de conexión para permitir el acceso abierto, aprovechar AWS Identity and Access Management (IAM) para autorizar el acceso o utilizar claves API para medir el acceso a sus funciones de Lambda por parte de otros usuarios.</p> <p>Para ver un ejemplo de ejercicio de introducción, consulte <a href="#">Invocación de una función de Lambda mediante un punto de conexión de Amazon API Gateway</a>.</p> <p>Para obtener más información sobre Amazon API Gateway, consulte <a href="https://aws.amazon.com/api-gateway/">https://aws.amazon.com/api-gateway/</a>.</p>	09 de julio de 2015

Cambio	Descripción	Fecha
La consola de Lambda ahora ofrece esquemas para crear fácilmente funciones de Lambda y probarlas.	La consola de Lambda proporciona un conjunto de esquemas. Cada esquema ofrece una configuración de origen de eventos de muestra y código de muestra para la función de Lambda que puede utilizar para crear fácilmente aplicaciones basadas en Lambda. Ahora, todos los ejercicios de Introducción de Lambda utilizan esquemas. Para obtener más información, consulte <a href="#">???</a> .	09 de julio de 2015
Lambda ahora permite utilizar Java para crear funciones de Lambda.	A partir de ahora, puede escribir el código de Lambda en Java. Para obtener más información, consulte <a href="#">Creación de funciones de Lambda con Java</a> .	15 de junio de 2015
Lambda ahora permite especificar un objeto de Amazon S3 como archivo .zip de la función al crear o actualizar una función de Lambda.	Puede cargar el paquete de implementación (archivo.zip) de una función de Lambda en un bucket de Amazon S3 de la misma región en la que desea crear la función de Lambda. A continuación, puede especificar el nombre del bucket y un nombre de la clave de objeto al crear o actualizar una función de Lambda.	28 de mayo de 2015

Cambio	Descripción	Fecha
Lambda ahora está disponible con carácter general con soporte para backends para móviles	<p>Lambda ya se encuentra disponible con carácter general para su uso con fines de producción. La versión también presenta características nuevas que facilitan la creación de backends para móviles, tablets e Internet de las cosas (IoT) con Lambda que se escalan automáticamente sin necesidad de aprovisionar ni administrar infraestructura. Lambda ahora es compatible con eventos en tiempo real (síncronos) y asíncronos. Las características adicionales también incluyen una configuración y administración de orígenes de eventos más sencilla. El modelo de permisos y el modelo de programación se han simplificado mediante la introducción de políticas de recursos para las funciones de Lambda.</p> <p>La documentación se ha actualizado en consecuencia. Para obtener información, consulte los siguientes temas:</p> <p><a href="#">???</a></p> <p><a href="#">AWS Lambda</a></p>	9 de abril de 2015
Versión preliminar de	Versión de prueba de la Guía para desarrolladores de AWS Lambda.	13 de noviembre de 2014