



Entwicklerhandbuch für SDK Version 3

AWS SDK for JavaScript



AWS SDK for JavaScript: Entwicklerhandbuch für SDK Version 3

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

.....	ix
Was ist der AWS SDK for JavaScript?	1
Fangen Sie mit dem SDK an	1
Wartung und Support für SDK-Hauptversionen	2
Verwenden des SDKs mit Node.js	2
Verwenden des SDK mit AWS Cloud9	2
Verwenden des SDK mit AWS Amplify	2
Verwenden des SDK mit Webbrowsern	3
Verwendung von Browsern in V3	3
Häufige Anwendungsfälle	4
Informationen zu den Beispielen	4
Ressourcen	4
Erste Schritte	6
SDK-Authentifizierung mit AWS	6
Starten Sie eine AWS Access-Portal-Sitzung	8
Weitere Authentifizierungsinformationen	8
Beginnen Sie mit Node.js	9
Das Szenario	9
Voraussetzungen	9
Schritt 1: Richten Sie die Paketstruktur ein und installieren Sie die Client-Pakete	10
Schritt 2: Fügen Sie die erforderlichen Importe und den SDK-Code hinzu	11
Schritt 3: Führen Sie das Beispiel aus	13
Fangen Sie im Browser an	13
Das Szenario	14
Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle	14
Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu	15
Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu	16
Schritt 4: Richten Sie den Browsercode ein	17
Schritt 5: Führen Sie das Beispiel aus	18
Bereinigen	19
Einrichten des SDK für JavaScript	20
Voraussetzungen	20
Einrichten einer AWS Node.js-Umgebung	20
Unterstützte Webbrowser	21

Das SDK installieren	22
Laden des SDK	23
Konfigurieren Sie das SDK für JavaScript	24
Konfiguration pro Dienst	24
Legen Sie die Konfiguration pro Dienst fest	25
Legen Sie die Region fest AWS	25
In einem Client-Klassenkonstruktor	26
Verwenden Sie eine Umgebungsvariable	26
Verwenden Sie eine gemeinsam genutzte Konfigurationsdatei	26
Reihenfolge der Festlegung der Region	27
Legen Sie die Anmeldeinformationen fest	27
Bewährte Methoden für Anmeldeinformationen	27
Festlegen von Anmeldeinformationen in Node.js	28
Anmeldeinformationen in einem Webbrowser einrichten	32
Überlegungen zu Node.js	36
Verwenden Sie die integrierten Node.js -Module	36
Verwenden Sie npm-Pakete	37
MaxSockets in Node.js konfigurieren	37
Verbindungen mit Keep-Alive in Node.js wiederverwenden	38
Konfigurieren Sie Proxys für Node.js	39
Registrieren Sie Zertifikatspakete in Node.js	40
Überlegungen zum Browser-Skript	40
Erstellen Sie das SDK für Browser	41
Cross-Origin Resource Sharing (CORS)	41
Bundle mit Webpack	45
Arbeiten Sie mit AWS Diensten	50
Serviceobjekte erstellen und aufrufen	50
Geben Sie die Parameter des Serviceobjekts an	51
Rufen Sie Dienste asynchron auf	51
Asynchrone Aufrufe verwalten	52
Verwenden Sie async/await	53
Nutze Versprechen	54
Verwenden Sie eine Rückruffunktion	56
Erstellen Sie Service-Client-Anfragen	57
Bearbeiten Sie die Antworten der Servicekunden	58
In der Antwort zurückgegebene Zugriffsdaten	58

Auf Fehlerinformationen zugreifen	58
Arbeiten Sie mit JSON	59
JSON als Serviceobjektparameter	60
Teilmenge der Codebeispiele mit Anleitung	61
JavaScript ES6/CommonJS-Syntax	62
Beispiele für Amazon DynamoDB	65
Beispiele für AWS Elemental MediaConvert	90
Beispiele für AWS Lambda	113
Amazon Lex Beispiele	113
Beispiele für Amazon Polly	114
Beispiele für Amazon Redshift	118
Amazon-SES-Beispiele	126
Amazon SNS-Beispiele	154
Beispiele für Amazon Transcribe	190
Serviceübergreifend: Node.js auf einer Amazon EC2 EC2-Instance einrichten	202
Serviceübergreifend: App zum Senden von Daten	204
Serviceübergreifend: Transkriptions-App	212
Serviceübergreifend: Amazon API Gateway und Lambda	225
Serviceübergreifend: Serverlose Workflows mit Step Functions	240
Serviceübergreifend: Geplante Lambda-Ereignisse	256
Serviceübergreifendes Beispiel: Amazon Lex	268
Serviceübergreifend: Messaging-App	282
Verwenden von AWS Cloud9 mit dem SDK für JavaScript	296
Schritt 1: Einrichten Ihres - AWS Kontos zur Verwendung von AWS Cloud9	296
Schritt 2: Einrichten Ihrer AWS Cloud9 Entwicklungsumgebung	297
Schritt 3: Einrichten des SDK für JavaScript	297
So richten Sie das SDK für JavaScript für Node.js ein	297
So richten Sie das SDK für JavaScript im Browser ein	298
Schritt 4: Beispielcode herunterladen	298
Schritt 5: Beispielcode ausführen und debuggen	299
Codebeispiele	300
Aktionen und Szenarien	300
Auto Scaling	302
Amazon Bedrock	345
Amazon Bedrock Runtime	349
Agents für Amazon Bedrock	393

Agenten für Amazon Bedrock Runtime	407
CloudWatch	409
CloudWatch Ereignisse	425
CloudWatch Logs	432
CodeBuild	450
Amazon Cognito Identity Provider	453
Amazon DocumentDB	473
DynamoDB	475
Amazon EC2	534
Elastic Load Balancing — Version 2	617
EventBridge	666
AWS Glue	674
HealthImaging	699
IAM	759
Kinesis	870
Lambda	875
Amazon Personalize	901
Amazon Personalize Events	917
Amazon Personalize Runtime	921
Amazon Pinpoint	925
Amazon-Redshift	935
Amazon S3	940
S3 Glacier	1012
SageMaker	1016
Secrets Manager	1055
Amazon SES	1056
Amazon SNS	1080
Amazon SQS	1118
Step Functions	1158
AWS STS	1159
AWS Support	1162
Amazon Transcribe	1181
Serviceübergreifende Beispiele	1190
Eine Amazon-Transcribe-App entwickeln	1191
Erstellen einer Amazon-Transcribe-Streaming-App	1191
Erstellen Sie eine App zum Senden von Daten an eine DynamoDB-Tabelle	1192

Einen Amazon Lex Lex-Chatbot erstellen	1192
Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos	1193
Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten	1194
Erstellen Sie einen Tracker für Aurora-Serverless-Arbeitsaufgaben	1194
Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung	1195
Erstellen einer Anwendung zum Analysieren von Kundenfeedback	1195
Erkennen von PSA in Bildern	1199
Erkennen von Objekten in Bildern	1200
Erkennen Sie Personen und Objekte in einem Video	1201
Aufrufen einer Lambda-Funktion von einem Browser aus	1201
Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion	1202
Verwenden von Step Functions, um Lambda-Funktionen aufzurufen	1203
Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion	1203
Sicherheit	1205
Datenschutz	1206
Identitäts- und Zugriffsverwaltung	1207
Zielgruppe	1207
Authentifizierung mit Identitäten	1208
Verwalten des Zugriffs mit Richtlinien	1212
Wie AWS-Services arbeiten Sie mit IAM	1215
Fehlerbehebung bei AWS Identität und Zugriff	1215
Compliance-Validierung	1217
Ausfallsicherheit	1218
Sicherheit der Infrastruktur	1219
Erzwingen Sie eine TLS-Mindestversion	1220
Überprüfen und Erzwingen von TLS in Node.js	1220
Überprüfen und Erzwingen von TLS in einem Browserskript	1223
Migrieren Sie zu v3	1225
Migrieren Sie mit Codemod zu v3	1225
Verwenden Sie Codemod, um vorhandenen v2-Code zu migrieren	1225
Was ist neu in Version 3	1226
Modularisierte Pakete	1227
Codegröße vergleichen	1228
Befehle in Version 3 aufrufen	1229
Neuer Middleware-Stack	1231
Was ist der Unterschied zwischen der Version 2 und der Version 3	1232

Konstrukteure des Kunden	1233
Anbieter von Anmeldeinformationen	1238
Überlegungen zu Amazon S3	1245
DynamoDB-Dokumentenclient	1246
Kellner und Unterzeichner	1248
Hinweise zu bestimmten Servicekunden	1249
Dokumentverlauf	1253
Dokumentverlauf	1253

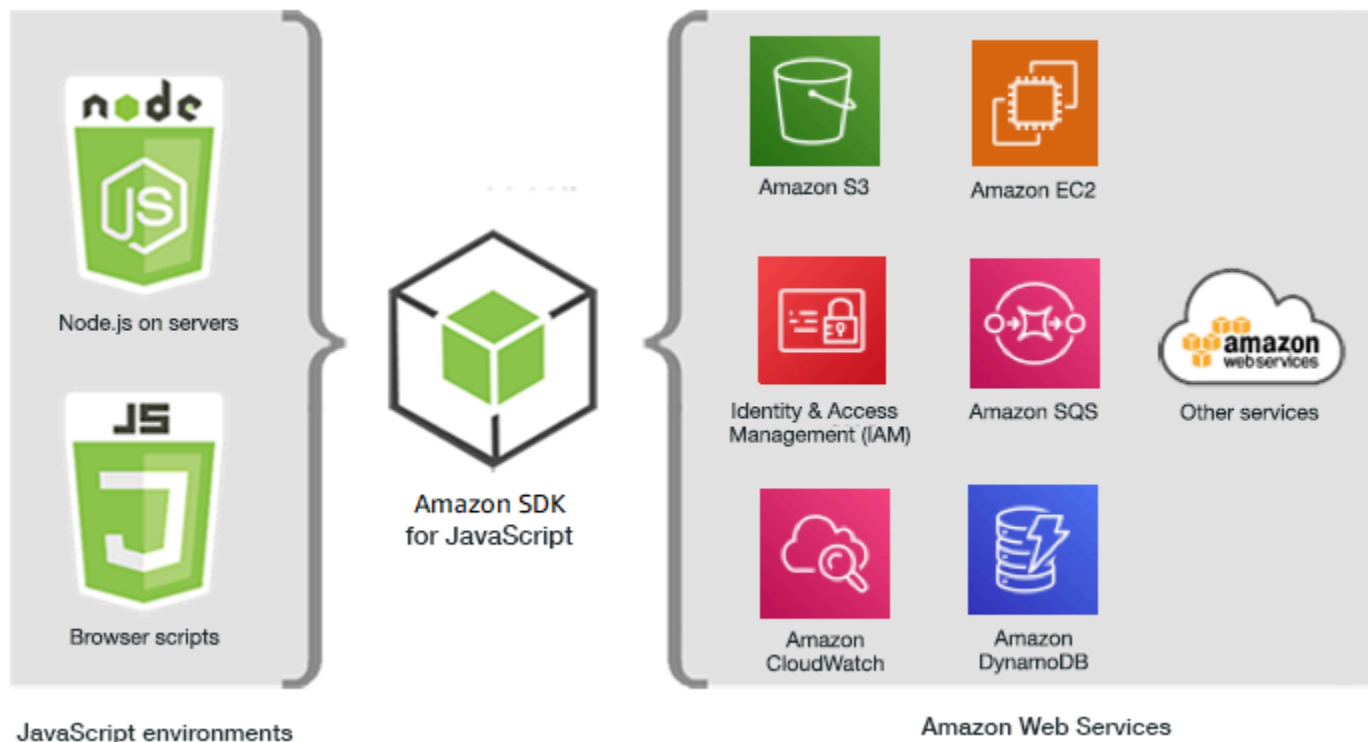
Das [AWS SDK for JavaScript V3-API-Referenzhandbuch](#) beschreibt detailliert alle API-Operationen für die AWS SDK for JavaScript Version 3 (V3).

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.

Was ist der AWS SDK for JavaScript?

Willkommen im AWS SDK for JavaScript Entwicklerhandbuch. Dieses Handbuch enthält allgemeine Informationen zur Einrichtung und Konfiguration von AWS SDK for JavaScript. Es führt Sie auch durch Beispiele und Anleitungen zum Ausführen verschiedener AWS Dienste mit dem AWS SDK for JavaScript.

Das [AWS SDK for JavaScript v3-API-Referenzhandbuch](#) enthält eine JavaScript API für AWS Dienste. Sie können die JavaScript API verwenden, um Bibliotheken oder Anwendungen für [Node.js](#) oder den Browser zu erstellen.



Fangen Sie mit dem SDK an

Wenn Sie bereit sind, das SDK in der Praxis auszuprobieren, folgen Sie den Beispielen unter [Erste Schritte](#).

Informationen zum Einrichten Ihrer Entwicklungsumgebung finden Sie unter [Einrichten des SDK für JavaScript](#).

Wenn Sie derzeit Version 2.x des SDK für verwenden JavaScript, finden Sie spezifische Anleitungen unter [Migration zu Version 3](#).

Wenn Sie nach Codebeispielen für suchen AWS-Services, finden Sie unter [SDK für JavaScript \(v3\) - Codebeispiele](#).

Wartung und Support für SDK-Hauptversionen

Informationen zu Wartung und Support für SDK-Hauptversionen und deren zugrunde liegende Abhängigkeiten finden Sie im [AWS -Referenzhandbuch zu SDKs und Tools](#):

- [AWS Wartungsrichtlinie für SDKs und Tools](#)
- [AWS Matrix zur Unterstützung der Versionen von SDKs und Tools](#)

Verwenden des SDKs mit Node.js

Node.js ist eine plattformübergreifende Laufzeit für die Ausführung serverseitiger JavaScript Anwendungen. Sie können Node.js auf einer Amazon Elastic Compute Cloud (Amazon EC2) - Instance einrichten, um sie auf einem Server auszuführen. Sie können Node.js auch verwenden, um AWS Lambda On-Demand-Funktionen zu schreiben.

Die Verwendung des SDK für Node.js unterscheidet sich von der Art und Weise, wie Sie es JavaScript in einem Webbrowser verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs zwischen Node.js und dem Browser unterscheidet, weisen wir auf diese Unterschiede hin.

Verwenden des SDK mit AWS Cloud9

Sie können Node.js -Anwendungen auch mithilfe des SDK für JavaScript in der AWS Cloud9 IDE entwickeln. Weitere Hinweise zur Verwendung AWS Cloud9 mit dem SDK für JavaScript finden Sie unter [Verwenden von AWS Cloud9 mit der AWS SDK for JavaScript](#).

Verwenden des SDK mit AWS Amplify

Für browserbasierte Web-, Mobil- und Hybrid-Apps können Sie die [AWS Amplify Bibliothek auch auf GitHub](#) verwenden. Es erweitert das SDK für JavaScript und bietet eine deklarative Schnittstelle.

Note

Frameworks wie Amplify bieten möglicherweise nicht die gleiche Browserunterstützung wie das SDK für JavaScript. Einzelheiten finden Sie in der Dokumentation des Frameworks.

Verwenden des SDK mit Webbrowsern

Alle gängigen Webbrowser unterstützen die Ausführung von JavaScript. JavaScript Code, der in einem Webbrowser ausgeführt wird, wird oft als JavaScriptclientseitig bezeichnet.

Eine Liste der Browser, die von der unterstützt werden AWS SDK for JavaScript, finden Sie unter [Unterstützte Webbrowser](#)

Die Verwendung des SDK für JavaScript in einem Webbrowser unterscheidet sich von der Art und Weise, wie Sie es für Node.js verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs zwischen Node.js und dem Browser unterscheidet, weisen wir auf diese Unterschiede hin.

Verwendung von Browsern in V3

V3 ermöglicht es Ihnen, nur das SDK für JavaScript Dateien, die Sie benötigen, zu bündeln und in den Browser aufzunehmen, wodurch der Overhead reduziert wird.

Um Version 3 des SDK for JavaScript in Ihren HTML-Seiten zu verwenden, müssen Sie die erforderlichen Client-Module und alle erforderlichen JavaScript Funktionen mithilfe von Webpack in einer einzigen JavaScript Datei bündeln und diese in einem Skript-Tag auf Ihren HTML-Seiten hinzufügen. <head> Beispielsweise:

```
<script src="./main.js"></script>
```

Note

Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#)

Um Version 2 des SDK für zu verwenden JavaScript, fügen Sie stattdessen ein Skript-Tag hinzu, das auf die neueste Version des V2-SDK verweist. Weitere Informationen finden Sie im [Beispiel](#) im AWS SDK for JavaScript Developer Guide v2.

Häufige Anwendungsfälle

Die Verwendung des SDK für JavaScript in Browserskripten ermöglicht die Realisierung einer Reihe überzeugender Anwendungsfälle. Im Folgenden finden Sie einige Ideen für Dinge, die Sie in einer Browseranwendung erstellen können, indem Sie das SDK für den JavaScript Zugriff auf verschiedene Webdienste verwenden.

- Erstellen Sie eine benutzerdefinierte Konsole für AWS Dienste, in der Sie auf Funktionen aus verschiedenen Regionen und Diensten zugreifen und diese kombinieren können, um Ihre Organisations- oder Projektanforderungen bestmöglich zu erfüllen.
- Verwenden Sie Amazon Cognito Identity, um authentifizierten Benutzerzugriff auf Ihre Browseranwendungen und Websites zu ermöglichen, einschließlich der Verwendung der Drittanbieter-Authentifizierung von Facebook und anderen.
- Verwenden Sie Amazon Kinesis, um Klickstreams oder andere Marketingdaten in Echtzeit zu verarbeiten.
- Verwenden Sie Amazon DynamoDB für die serverlose Datenpersistenz, z. B. für individuelle Benutzereinstellungen für Website-Besucher oder Anwendungsbenutzer.
- Wird verwendet AWS Lambda , um proprietäre Logik zu kapseln, die Sie aus Browser-Skripten aufrufen können, ohne Ihr geistiges Eigentum herunterzuladen und den Benutzern zugänglich zu machen.

Informationen zu den Beispielen

Sie können das SDK im [AWS Code](#) Example JavaScript Repository nach Beispielen durchsuchen.

Ressourcen

Zusätzlich zu diesem Handbuch sind die folgenden Online-Ressourcen für SDK für JavaScript Entwickler verfügbar:

- [AWS SDK for JavaScript V3-API-Referenzhandbuch](#)
- [AWS Referenzhandbuch für SDKs und Tools](#): Enthält Einstellungen, Funktionen und andere grundlegende Konzepte, die bei AWS SDKs üblich sind.

- [JavaScript Blog für Entwickler](#)
- [AWS JavaScript Forum](#)
- [JavaScript Beispiele im AWS Code-Katalog](#)
- [AWS Code-Beispiel-Repository](#)
- [Gitter-Kanal](#)
- [Stack-Überlauf](#)
- [Fragen zu Stack Overflow TaggeAWS -sdk-js](#)
- GitHub
 - [SDK-Quelle](#)
 - [Quelle der Dokumentation](#)

Fangen Sie an mit dem AWS SDK for JavaScript

Das AWS SDK for JavaScript ermöglicht den Zugriff auf Webdienste entweder in einer Browser- oder in einer Node.js -Umgebung. In diesem Abschnitt finden Sie Übungen für die ersten Schritte, die Ihnen zeigen, wie Sie mit dem SDK für JavaScript jede dieser JavaScript Umgebungen arbeiten.

Note

Sie können Node.js -Anwendungen entwickeln und JavaScript für browserbasierte Anwendungen das SDK JavaScript in der AWS Cloud9 IDE verwenden. Ein Beispiel zur Verwendung AWS Cloud9 für die Entwicklung von Node.js finden Sie unter [Verwenden von AWS Cloud9 mit der AWS SDK for JavaScript](#).

Themen

- [SDK-Authentifizierung mit AWS](#)
- [Beginnen Sie mit Node.js](#)
- [Fangen Sie im Browser an](#)

SDK-Authentifizierung mit AWS

Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit AWS-Services authentifiziert. Sie können den programmatischen Zugriff auf AWS Ressourcen je nach Umgebung und verfügbarem AWS Zugriff auf unterschiedliche Weise konfigurieren.

Informationen zur Auswahl Ihrer Authentifizierungsmethode und deren Konfiguration für das SDK finden Sie unter [Authentifizierung und Zugriff](#) im Referenzhandbuch für AWS SDKs und Tools.

Wir empfehlen neuen Benutzern, die sich lokal weiterentwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode zur Einrichtung AWS IAM Identity Center erhalten. Diese Methode beinhaltet die Installation von, AWS CLI um die Konfiguration zu vereinfachen und sich regelmäßig beim AWS Zugangsportale anzumelden. Wenn Sie sich für diese Methode entscheiden, sollte Ihre Umgebung die folgenden Elemente enthalten, nachdem Sie das Verfahren zur [IAM Identity Center-Authentifizierung](#) im Referenzhandbuch für AWS SDKs und Tools abgeschlossen haben:

- Die AWS CLI, mit der Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.

- Eine [gemeinsam genutzte AWSconfig Datei](#) mit einem [default] Profil mit einer Reihe von Konfigurationswerten, auf die vom SDK aus verwiesen werden kann. Den Speicherort dieser Datei finden Sie unter [Speicherort der freigegebenen Dateien](#) im Referenzhandbuch für AWS SDKs und Tools.
- Die gemeinsam genutzte config Datei legt die [region](#)Einstellung fest. Dies legt die Standardeinstellung AWS-Region fest, die das SDK für AWS Anfragen verwendet. Diese Region wird für SDK-Dienstanforderungen verwendet, für die keine zu verwendende Region angegeben ist.
- Das SDK verwendet die [Konfiguration des SSO-Token-Anbieters](#) des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet AWS werden. Der sso_role_name Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM Identity Center-Berechtigungssatz verbunden ist, ermöglicht den Zugriff auf die in Ihrer AWS-Services Anwendung verwendeten.

Die folgende config Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die sso_session Einstellung des Profils bezieht sich auf den benannten [sso-sessionAbschnitt](#). Der sso-session Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Für AWS SDK for JavaScript Version 3 müssen Ihrer Anwendung keine zusätzlichen Pakete (wie SSO undSSO0IDC) hinzugefügt werden, um die IAM Identity Center-Authentifizierung zu verwenden.

Einzelheiten zur expliziten Verwendung dieses Anmeldeinformationsanbieters finden Sie [fromSSO\(\)](#) auf der npm-Website (Node.js package manager).

Starten Sie eine AWS Access-Portal-Sitzung

Bevor Sie eine Zugriffsanwendung ausführen AWS-Services, benötigen Sie eine aktive AWS Access-Portal-Sitzung, damit das SDK die IAM Identity Center-Authentifizierung zur Auflösung von Anmeldeinformationen verwenden kann. Abhängig von Ihrer konfigurierten Sitzungsdauer läuft Ihr Zugriff irgendwann ab und das SDK wird auf einen Authentifizierungsfehler stoßen. Um sich beim AWS Zugriffportal anzumelden, führen Sie den folgenden Befehl in der aus AWS CLI.

```
aws sso login
```

Wenn Sie die Anweisungen befolgt haben und ein Standardprofil eingerichtet haben, müssen Sie den Befehl nicht mit einer `--profile` Option aufrufen. Wenn die Konfiguration Ihres SSO-Token-Anbieters ein benanntes Profil verwendet, lautet der Befehl `aws sso login --profile named-profile`.

Führen Sie den folgenden AWS CLI Befehl aus, um optional zu testen, ob Sie bereits eine aktive Sitzung haben.

```
aws sts get-caller-identity
```

Wenn Ihre Sitzung aktiv ist, werden in der Antwort auf diesen Befehl das in der gemeinsam genutzten `config` Datei konfigurierte IAM Identity Center-Konto und der Berechtigungssatz gemeldet.

Note

Wenn Sie bereits eine aktive AWS Access-Portal-Sitzung haben und ausführen `aws sso login`, müssen Sie keine Anmeldeinformationen angeben.

Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Weitere Authentifizierungsinformationen

Menschliche Benutzer, auch bekannt als menschliche Identitäten, sind die Personen, Administratoren, Entwickler, Betreiber und Verbraucher Ihrer Anwendungen. Sie müssen über eine Identität verfügen, um auf Ihre AWS Umgebungen und Anwendungen zugreifen zu können.

Menschliche Benutzer, die Mitglieder Ihres Unternehmens sind, also Sie, der Entwickler, werden als Personalidentitäten bezeichnet.

Verwenden Sie beim Zugriff AWS temporäre Anmeldeinformationen. Sie können einen Identitätsanbieter für Ihre menschlichen Benutzer verwenden, um Verbundzugriff auf AWS Konten zu ermöglichen, indem Sie Rollen übernehmen, die temporäre Anmeldeinformationen bereitstellen. Für eine zentralisierte Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center (IAM Identity Center) zu verwenden, um den Zugriff auf Ihre Konten und die Berechtigungen innerhalb dieser Konten zu verwalten. Weitere Alternativen finden Sie im Folgenden:

- Weitere Informationen zu bewährten Methoden finden Sie unter [Bewährte Methoden für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.
- Informationen zum Erstellen kurzfristiger AWS Anmeldeinformationen finden Sie unter [Temporäre Sicherheitsanmeldeinformationen](#) im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen Anbietern von AWS SDK for JavaScript V3-Anmeldeinformationen finden Sie unter [Standardisierte Anbieter von Anmeldeinformationen im Referenzhandbuch](#) für AWS SDKs und Tools.

Beginnen Sie mit Node.js

Diese Anleitung zeigt Ihnen, wie Sie ein NPM-Paket initialisieren, Ihrem Paket einen Service-Client hinzufügen und das JavaScript SDK verwenden, um eine Serviceaktion aufzurufen.

Das Szenario

Erstellen Sie ein neues NPM-Paket mit einer Hauptdatei, die Folgendes tut:

- Erstellt einen Amazon Simple Storage Service-Bucket
- Fügt ein Objekt in den Amazon S3 S3-Bucket
- Liest das Objekt im Amazon S3 S3-Bucket
- Bestätigt, ob der Benutzer Ressourcen löschen möchte

Voraussetzungen

Bevor Sie das Beispiel ausführen können, müssen Sie Folgendes tun:

- Konfigurieren Sie Ihre SDK-Authentifizierung. Weitere Informationen finden Sie unter [SDK-Authentifizierung mit AWS](#).
- Installieren Sie [Node.js](#).

Schritt 1: Richten Sie die Paketstruktur ein und installieren Sie die Client-Pakete

So richten Sie die Paketstruktur ein und installieren die Client-Pakete:

1. Erstellen Sie einen neuen Ordner `nodegetstarted`, der das Paket enthalten soll.
2. Navigieren Sie von der Befehlszeile aus zu dem neuen Ordner.
3. Führen Sie den folgenden Befehl aus, um eine `package.json` Standarddatei zu erstellen:

```
npm init -y
```

4. Führen Sie den folgenden Befehl aus, um das Amazon S3 S3-Client-Paket zu installieren:

```
npm i @aws-sdk/client-s3
```

5. `"type": "module"` Zur `package.json` Datei hinzufügen. Dadurch wird Node.js angewiesen, die moderne ESM-Syntax zu verwenden. Das finale `package.json` sollte in etwa wie folgt aussehen:

```
{
  "name": "example-javascriptv3-get-started-node",
  "version": "1.0.0",
  "description": "This guide shows you how to initialize an NPM package, add a service client to your package, and use the JavaScript SDK to call a service action.",
  "main": "index.js",
  "scripts": {
    "test": "vitest run **/*.unit.test.js"
  },
  "author": "Your Name",
  "license": "Apache-2.0",
  "dependencies": {
    "@aws-sdk/client-s3": "^3.420.0"
  },
}
```

```
"type": "module"  
}
```

Schritt 2: Fügen Sie die erforderlichen Importe und den SDK-Code hinzu

Fügen Sie den folgenden Code zu einer Datei hinzu, die `index.js` im `nodegetstarted` Ordner benannt ist.

```
// This is used for getting user input.  
import { createInterface } from "readline/promises";  
  
import {  
  S3Client,  
  PutObjectCommand,  
  CreateBucketCommand,  
  DeleteObjectCommand,  
  DeleteBucketCommand,  
  paginateListObjectsV2,  
  GetObjectCommand,  
} from "@aws-sdk/client-s3";  
  
export async function main() {  
  // A region and credentials can be declared explicitly. For example  
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would  
  // initialize the client with those settings. However, the SDK will  
  // use your local configuration and credentials if those properties  
  // are not defined here.  
  const s3Client = new S3Client({});  
  
  // Create an Amazon S3 bucket. The epoch timestamp is appended  
  // to the name to make it unique.  
  const bucketName = `test-bucket-${Date.now()}`;  
  await s3Client.send(  
    new CreateBucketCommand({  
      Bucket: bucketName,  
    })  
  );  
  
  // Put an object into an Amazon S3 bucket.  
  await s3Client.send(  
    new PutObjectCommand({
```

```
    Bucket: bucketName,
    Key: "my-first-object.txt",
    Body: "Hello JavaScript SDK!",
  })
);

// Read the object.
const { Body } = await s3Client.send(
  new GetObjectCommand({
    Bucket: bucketName,
    Key: "my-first-object.txt",
  })
);

console.log(await Body.transformToString());

// Confirm resource deletion.
const prompt = createInterface({
  input: process.stdin,
  output: process.stdout,
});

const result = await prompt.question("Empty and delete bucket? (y/n) ");
prompt.close();

if (result === "y") {
  // Create an async iterator over lists of objects in a bucket.
  const paginator = paginateListObjectsV2(
    { client: s3Client },
    { Bucket: bucketName }
  );
  for await (const page of paginator) {
    const objects = page.Contents;
    if (objects) {
      // For every object in each page, delete it.
      for (const object of objects) {
        await s3Client.send(
          new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key })
        );
      }
    }
  }

  // Once all the objects are gone, the bucket can be deleted.
}
```

```
    await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
  }
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

Den Beispielcode finden Sie [hier auf GitHub](#).

Schritt 3: Führen Sie das Beispiel aus

Note

Denken Sie daran, sich anzumelden! Wenn Sie IAM Identity Center zur Authentifizierung verwenden, denken Sie daran, sich mit dem AWS CLI `aws sso login` folgenden Befehl anzumelden.

1. Führen Sie `node index.js`.
2. Wählen Sie aus, ob der Bucket geleert und gelöscht werden soll.
3. Wenn Sie den Bucket nicht löschen, stellen Sie sicher, dass Sie ihn manuell leeren und später löschen.

Fangen Sie im Browser an

Dieser Abschnitt führt Sie durch ein Beispiel, das zeigt, wie Sie Version 3 (V3) des SDK für JavaScript im Browser ausführen.

Note

Die Ausführung von V3 im Browser unterscheidet sich geringfügig von Version 2 (V2). Weitere Informationen finden Sie unter [Verwendung von Browsern in V3](#).

Weitere Beispiele für die Verwendung von (V3) des SDK für JavaScript finden Sie unter [SDK für JavaScript \(v3\) -Codebeispiele](#).

Dieses Beispiel für eine Webanwendung zeigt Ihnen:

- So greifen Sie mit Amazon Cognito zur Authentifizierung auf AWS Dienste zu.
- Wie liest man eine Liste von Objekten in einem Amazon Simple Storage Service (Amazon S3) - Bucket mithilfe einer AWS Identity and Access Management (IAM) -Rolle.

Note

Dieses Beispiel wird nicht AWS IAM Identity Center für die Authentifizierung verwendet.

Das Szenario

Amazon S3 ist ein Objektspeicherservice, der branchenführende Skalierbarkeit, Datenverfügbarkeit, Sicherheit und Leistung bietet. Sie können Amazon S3 verwenden, um Daten als Objekte in Containern, sogenannten Buckets, zu speichern. Weitere Informationen zu Amazon S3 finden Sie im [Amazon S3 S3-Benutzerhandbuch](#).

Dieses Beispiel zeigt Ihnen, wie Sie eine Web-App einrichten und ausführen, die eine IAM-Rolle zum Lesen aus einem Amazon S3 S3-Bucket annimmt. Das Beispiel verwendet die React-Frontend-Bibliothek und die Vite-Frontend-Tools, um eine Entwicklungsumgebung bereitzustellen. JavaScript Die Web-App verwendet einen Amazon Cognito Cognito-Identitätspool, um Anmeldeinformationen bereitzustellen, die für den Zugriff auf AWS Dienste erforderlich sind. Das mitgelieferte Codebeispiel demonstriert die grundlegenden Muster für das Laden und Verwenden des SDK JavaScript in Web-Apps.

Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle

In dieser Übung erstellen und verwenden Sie einen Amazon Cognito Cognito-Identitätspool, um nicht authentifizierten Zugriff auf Ihre Web-App für den Amazon S3 S3-Service bereitzustellen. Durch die Erstellung eines Identitätspools wird auch eine AWS Identity and Access Management (IAM-) Rolle zur Unterstützung nicht authentifizierter Gastbenutzer erstellt. In diesem Beispiel werden wir nur mit der Rolle „Nicht authentifizierter Benutzer“ arbeiten, damit wir uns auf die Aufgabe konzentrieren

können. Sie können die Unterstützung für einen Identitätsanbieter und authentifizierte Benutzer zu einem späteren Zeitpunkt integrieren. Weitere Informationen zum Hinzufügen eines Amazon Cognito-Identitätspools finden Sie unter [Tutorial: Creating an Identity Pool](#) im Amazon Cognito Developer Guide.

So erstellen Sie einen Amazon Cognito Cognito-Identitätspool und die zugehörige IAM-Rolle

1. Melden Sie sich bei der Amazon Cognito Cognito-Konsole an AWS Management Console und öffnen Sie sie unter <https://console.aws.amazon.com/cognito/>.
2. Wählen Sie im linken Navigationsbereich Identity Pools aus.
3. Wählen Sie Identitätspool erstellen.
4. Wählen Sie unter Vertrauensstellung im Identitätspool konfigurieren die Option Gastzugriff für die Benutzerauthentifizierung aus.
5. Wählen Sie unter Berechtigungen konfigurieren die Option Neue IAM-Rolle erstellen aus und geben Sie einen Namen (z. B. get StartedRole) in den IAM-Rollennamen ein.
6. Geben Sie unter Eigenschaften konfigurieren einen Namen (z. B. get StartedPool) in das Feld Identitätspoolname ein.
7. Bestätigen Sie unter Überprüfen und erstellen die Auswahl, die Sie für Ihren neuen Identitätspool getroffen haben. Wählen Sie Bearbeiten, um zum Assistenten zurückzukehren und Einstellungen zu ändern. Wählen Sie danach Identitätspool erstellen aus.
8. Notieren Sie sich die Identitätspool-ID und die Region des neu erstellten Amazon Cognito Cognito-Identitätspools. *Sie benötigen diese Werte, um IDENTITY_POOL_ID und REGION in zu ersetzen.* [Schritt 4: Richten Sie den Browsercode ein](#)


Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt haben, können Sie Berechtigungen für Amazon S3 hinzufügen, die von Ihrer Web-App benötigt werden.

Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu

Um den Zugriff auf einen Amazon S3 S3-Bucket in Ihrer Web-App zu aktivieren, verwenden Sie die nicht authentifizierte IAM-Rolle (z. B. get StartedRole), die für Ihren Amazon Cognito Cognito-Identitätspool erstellt wurde (z. B. get). StartedPool Dazu müssen Sie der Rolle eine IAM-Richtlinie hinzufügen. Weitere Informationen zum Ändern von IAM-Rollen finden Sie unter [Ändern einer Rollenberechtigungsrichtlinie](#) im IAM-Benutzerhandbuch.

So fügen Sie der IAM-Rolle, die nicht authentifizierten Benutzern zugeordnet ist, eine Amazon S3 S3-Richtlinie hinzu

1. [Melden Sie sich bei der an AWS Management Console und öffnen Sie die IAM-Konsole unter `https://console.aws.amazon.com/iam/`.](https://console.aws.amazon.com/iam/)
2. Wählen Sie im linken Navigationsbereich Roles aus.
3. Wählen Sie den Namen der Rolle aus, die Sie ändern möchten (z. B. get StartedRole), und wählen Sie dann den Tab Permissions.
4. Wählen Sie „Berechtigungen hinzufügen“ und anschließend „Richtlinien anhängen“.
5. Suchen Sie auf der Seite „Berechtigungen hinzufügen“ für diese Rolle nach dem Kontrollkästchen für AmazonS3 Access und aktivieren Sie es ReadOnly.

 Note

Sie können diesen Prozess verwenden, um den Zugriff auf jeden AWS Dienst zu aktivieren.

6. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt und Ihrer IAM-Rolle für nicht authentifizierte Benutzer Berechtigungen für Amazon S3 hinzugefügt haben, können Sie einen Amazon S3 S3-Bucket hinzufügen und konfigurieren.

Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu

In diesem Schritt fügen Sie einen Amazon S3 S3-Bucket und ein Objekt für das Beispiel hinzu. Sie werden auch Cross-Origin Resource Sharing (CORS) für den Bucket aktivieren. Weitere Informationen zum Erstellen von Amazon S3-Buckets und -Objekten finden Sie unter [Erste Schritte mit Amazon S3](#) im Amazon S3 S3-Benutzerhandbuch.

Um einen Amazon S3 S3-Bucket und ein Objekt mit CORS hinzuzufügen

1. Melden Sie sich bei der Amazon S3 S3-Konsole an AWS Management Console und öffnen Sie sie unter <https://console.aws.amazon.com/s3/>.
2. Wählen Sie im linken Navigationsbereich Buckets und dann Create Bucket aus.
3. Geben Sie einen Bucket-Namen ein, der den [Benennungsregeln für Buckets entspricht \(z. B. `getstartedbucket`\)](#), und wählen Sie Create Bucket aus.

4. Wählen Sie den Bucket aus, den Sie erstellt haben, und klicken Sie dann auf den Tab Objekte. Klicken Sie anschließend auf Upload.
5. Wählen Sie unter Files and folders (Dateien und Ordner) die Option Add files (Dateien hinzufügen) aus.
6. Wählen Sie eine hochzuladende Datei und dann Öffnen aus. Wählen Sie dann Hochladen, um den Upload des Objekts in Ihren Bucket abzuschließen.
7. Wählen Sie als Nächstes den Tab „Berechtigungen“ Ihres Buckets und anschließend im Abschnitt Cross-Origin Resource Sharing (CORS) die Option Bearbeiten aus. Geben Sie den folgenden JSON-Code ein:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

8. Wählen Sie Änderungen speichern aus.

Nachdem Sie einen Amazon S3 S3-Bucket und ein Objekt hinzugefügt haben, können Sie den Browsercode einrichten.

Schritt 4: Richten Sie den Browsercode ein

Die Beispielanwendung besteht aus einer einseitigen React-Anwendung. Die Dateien für dieses Beispiel finden Sie [hier auf GitHub](#).

Um die Beispielanwendung einzurichten

1. Installieren Sie [Node.js](#).
2. Klonen Sie das [AWS Codebeispiel-Repository](#) von der Befehlszeile aus:

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. Navigieren Sie zur Beispielanwendung:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. Führen Sie den folgenden Befehl aus, um die erforderlichen Pakete zu installieren:

```
npm install
```

5. Öffnen Sie anschließend `src/App.tsx` in einem Texteditor und gehen Sie wie folgt vor:

- Ersetzen Sie *YOUR_IDENTITY_POOL_ID* durch die *Amazon Cognito Cognito-Identitätspool-ID*, die Sie notiert haben. [Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle](#)
- Ersetzen Sie den Wert für Region durch die Region, die Ihrem Amazon S3-Bucket und Amazon Cognito Cognito-Identitätspool zugewiesen wurde. Beachten Sie, dass die Regionen für beide Dienste identisch sein müssen (z. B. us-east-2).
- Ersetzen Sie den *Bucket-Namen* durch den *Bucket-Namen*, den Sie in erstellt haben. [Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu](#)

Nachdem Sie den Text ersetzt haben, speichern Sie die `App.tsx` Datei. Sie sind jetzt bereit, die Web-App auszuführen.

Schritt 5: Führen Sie das Beispiel aus

Um die Beispielanwendung auszuführen

1. Navigieren Sie von der Befehlszeile aus zur Beispielanwendung:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. Führen Sie in der Befehlszeile den folgenden Befehl aus:

```
npm run dev
```

Die Vite-Entwicklungsumgebung wird mit der folgenden Meldung ausgeführt:

```
VITE v4.3.9 ready in 280 ms

# Local:   http://localhost:5173/
# Network: use --host to expose
# press h to show help
```

3. Navigieren Sie in Ihrem Webbrowser zu der oben angegebenen URL (z. B. <http://localhost:5173/>). Die Beispiel-App zeigt Ihnen eine Liste von Objektdateinamen in Ihrem Amazon S3 S3-Bucket.

Bereinigen

Gehen Sie wie folgt vor, um die Ressourcen zu bereinigen, die Sie in diesem Tutorial erstellt haben:

- Löschen Sie in [der Amazon S3 S3-Konsole](#) alle Objekte und erstellten Buckets (z. B. `getstartedbucket`).
- Löschen Sie in [der IAM-Konsole](#) den Rollennamen (z. B. `StartedRole`).
- Löschen Sie in [der Amazon Cognito Cognito-Konsole](#) den Namen des Identitätspools (z. B. `getStartedPool`).

Einrichten des SDK für JavaScript

In den Themen in diesem Abschnitt wird erläutert, wie Sie das SDK für installieren und laden, JavaScript damit Sie auf die vom SDK unterstützten Webservices zugreifen können.

Note

React-Native-Entwickler sollten verwenden AWS Amplify , um neue Projekte auf zu erstellen AWS. Weitere Informationen finden Sie im [aws-sdk-react-native](#) Archiv.

Themen

- [Voraussetzungen](#)
- [Installieren des SDK für JavaScript](#)
- [Laden des SDK für JavaScript](#)

Voraussetzungen

Installieren Sie Node.js auf Ihren Servern, sofern es noch nicht installiert ist.

Themen

- [Einrichten einer AWS Node.js-Umgebung](#)
- [Unterstützte Webbrowser](#)

Einrichten einer AWS Node.js-Umgebung

Verwenden Sie eine der folgenden Methoden, um eine AWS Node.js-Umgebung einzurichten, in der Sie Ihre Anwendung ausführen können:

- Wählen Sie ein Amazon Machine Image (AMI) mit vorinstalliertem Node.js aus. Erstellen Sie dann eine Amazon EC2-Instance mit diesem AMI. Wählen Sie beim Erstellen Ihrer Amazon EC2-Instance Ihr AMI aus der aus AWS Marketplace. Suchen Sie nach AWS Marketplace Node.js und wählen Sie eine AMI-Option aus, die eine vorinstallierte Version von Node.js (32-Bit oder 64-Bit) enthält.

- Erstellen Sie eine Amazon EC2-Instance und installieren Sie Node.js darauf. Weitere Informationen zum Installieren von Node.js auf einer Amazon Linux-Instance finden Sie unter [Node.js auf einer Amazon EC2 EC2-Instance einrichten](#).
- Erstellen Sie eine Serverless-Umgebung mit AWS Lambda , um Node.js als Lambda-Funktion auszuführen. Weitere Informationen zur Verwendung von Node.js innerhalb einer Lambda-Funktion finden Sie unter [Programmiermodell \(Node.js\)](#) im AWS Lambda -Entwicklerhandbuch.
- Stellen Sie Ihre Node.js-Anwendung in bereit AWS Elastic Beanstalk. Weitere Informationen zur Verwendung von Node.js mit Elastic Beanstalk finden Sie unter [Bereitstellen von Node.js-Anwendungen in AWS Elastic Beanstalk](#) im AWS Elastic Beanstalk -Entwicklerhandbuch.
- Erstellen Sie einen Node.js-Anwendungsserver mit AWS OpsWorks. Weitere Informationen zur Verwendung von Node.js mit finden Sie AWS OpsWorks unter [Erstellen Ihres ersten Node.js-Stacks](#) im AWS OpsWorks -Benutzerhandbuch.

Unterstützte Webbrowser

unterstützt AWS SDK for JavaScript alle modernen Webbrowser.

In Version 3.183.0 oder höher JavaScript verwendet das SDK für ES2020-Artefakte, die die folgenden Mindestversionen unterstützen.

Browser	Version
Google Chrome	80.0+
Mozilla Firefox	80.0+
Oper	63.0+
Microsoft Edge	80.0+
Apple Safari	14.1+
Samsung Internet	12.0+

In Version 3.182.0 oder früher JavaScript verwendet das SDK für ES5-Artefakte, die die folgenden Mindestversionen unterstützen.

Browser	Version
Google Chrome	49,0+
Mozilla Firefox	45,0+
Oper	36.0+
Microsoft Edge	12.0+
Windows Internet Explorer	N/A
Apple Safari	9.0+
Android-Browser	76.0+
UC-Browser	12.12+
Samsung Internet	5.0+

Note

Frameworks wie bieten AWS Amplify möglicherweise nicht denselben Browser-Support wie das SDK für JavaScript. Einzelheiten finden Sie in der [AWS Amplify Dokumentation](#).

Installieren des SDK für JavaScript

Nicht alle Services sind sofort im SDK oder in allen AWS Regionen verfügbar.

Um einen Service über die AWS SDK for JavaScript mit [npm zu installieren](#), geben Sie den [Node.js-Paketmanager](#) an der Eingabeaufforderung den folgenden Befehl ein, wobei **SERVICE** der Name eines Services ist, z. B. s3.

```
npm install @aws-sdk/client-SERVICE
```

Eine vollständige Liste der AWS SDK for JavaScript Service-Client-Pakete finden Sie im [API AWS SDK for JavaScript -Referenzhandbuch zu](#) .

Laden des SDK für JavaScript

Nachdem Sie das SDK installiert haben, können Sie ein Clientpaket in Ihre Knotenanwendung mit `ladenimport`. Um beispielsweise den Amazon S3-Client und den Amazon S3-[ListBuckets](#)Befehl zu laden, verwenden Sie Folgendes.

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```


Konfigurieren Sie das SDK für JavaScript

Bevor Sie das SDK JavaScript zum Aufrufen von Webdiensten mithilfe der API verwenden, müssen Sie das SDK konfigurieren. Sie müssen mindestens Folgendes konfigurieren:

- Die AWS Region, in der Sie Dienste anfordern werden
- Wie authentifiziert sich Ihr Code mit AWS

Zusätzlich zu diesen Einstellungen müssen Sie möglicherweise auch Berechtigungen für Ihre AWS Ressourcen konfigurieren. Sie können beispielsweise den Zugriff auf einen Amazon S3 S3-Bucket oder den Lesezugriff auf eine Amazon DynamoDB-Tabelle einschränken.

Das [Referenzhandbuch für AWS SDKs und Tools](#) enthält auch Einstellungen, Funktionen und andere grundlegende Konzepte, die vielen SDKs gemeinsam sind. AWS

In den Themen dieses Abschnitts wird beschrieben, wie Sie das SDK JavaScript für Node.js und die JavaScript Ausführung in einem Webbrowser konfigurieren können.

Themen

- [Konfiguration pro Dienst](#)
- [Legen Sie die Region fest AWS](#)
- [Legen Sie die Anmeldeinformationen fest](#)
- [Überlegungen zu Node.js](#)
- [Überlegungen zum Browser-Skript](#)

Konfiguration pro Dienst

Sie können das SDK konfigurieren, indem Sie Konfigurationsinformationen an ein Serviceobjekt übergeben.

Die Konfiguration auf Dienstebene bietet umfassende Kontrolle über einzelne Dienste und ermöglicht es Ihnen, die Konfiguration einzelner Dienstobjekte zu aktualisieren, wenn Ihre Anforderungen von der Standardkonfiguration abweichen.

Note

In Version 2.x konnte die AWS SDK for JavaScript Dienstkonfiguration an einzelne Client-Konstrukturen übergeben werden. Diese Konfigurationen würden jedoch zunächst automatisch zu einer Kopie der globalen SDK-Konfiguration zusammengeführt. `AWS.config` Außerdem wird `AWS.config.update({/* params */})` nur die aktualisierte Konfiguration für Service-Clients aufgerufen, die nach dem Aktualisierungsauftrag instanziiert wurden, nicht für bestehende Clients.

Dieses Verhalten sorgte häufig für Verwirrung und erschwerte es, dem globalen Objekt eine Konfiguration hinzuzufügen, die sich auf vorwärtskompatible Weise nur auf eine Teilmenge von Service-Clients auswirkt. In Version 3 gibt es keine globale Konfiguration mehr, die vom SDK verwaltet wird. Die Konfiguration muss an jeden Service-Client übergeben werden, der instanziiert wird. Es ist immer noch möglich, dieselbe Konfiguration für mehrere Clients gemeinsam zu verwenden, aber diese Konfiguration wird nicht automatisch mit einem globalen Status zusammengeführt.

Legen Sie die Konfiguration pro Dienst fest

Auf jeden Dienst, den Sie im SDK verwenden, JavaScript wird über ein Dienstobjekt zugegriffen, das Teil der API für diesen Dienst ist. Um beispielsweise auf den Amazon S3 S3-Service zuzugreifen, erstellen Sie das Amazon S3 S3-Serviceobjekt. Sie können für einen bestimmten Service Konfigurationseinstellungen als Teil des Konstruktors für dieses Serviceobjekt definieren.

Wenn Sie beispielsweise auf Amazon EC2-Objekte in mehreren AWS Regionen zugreifen müssen, erstellen Sie ein Amazon EC2-Serviceobjekt für jede Region und legen Sie dann die Regionskonfiguration für jedes Serviceobjekt entsprechend fest.

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

Legen Sie die Region fest AWS

Eine AWS Region ist eine benannte Gruppe von AWS Ressourcen in demselben geografischen Gebiet. Ein Beispiel für eine Region ist `us-east-1` die Region USA Ost (Nord-Virginia). Sie geben eine Region an, wenn Sie einen Service-Client im SDK für erstellen, JavaScript sodass das SDK

auf den Service in dieser Region zugreift. Einige -Services werden nur in bestimmten Regionen angeboten.

Das SDK für wählt standardmäßig JavaScript keine Region aus. Sie können die AWS Region jedoch mithilfe einer Umgebungsvariablen oder einer gemeinsam genutzten `config` Konfigurationsdatei festlegen.

In einem Client-Klassenkonstruktor

Wenn Sie ein Serviceobjekt instanziiieren, können Sie die AWS Region für diese Ressource als Teil des Client-Klassenkonstruktors angeben, wie hier gezeigt.

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

Verwenden Sie eine Umgebungsvariable

Sie können die Region mithilfe der Umgebungsvariablen `AWS_REGION` festlegen. Wenn Sie diese Variable definieren, JavaScript liest das SDK für sie und verwendet sie.

Verwenden Sie eine gemeinsam genutzte Konfigurationsdatei

So wie Sie mit der Datei mit gemeinsam genutzten Anmeldeinformationen Anmeldeinformationen für das SDK speichern können, können Sie Ihre AWS Region und andere Konfigurationseinstellungen in einer gemeinsam genutzten Datei speichern, die `config` nach dem zu verwendenden SDK benannt ist. Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen wahrheitsgemäßen Wert gesetzt ist, sucht das SDK für beim Laden JavaScript automatisch nach einer `config` Datei. Wo Sie die `config`-Datei speichern, hängt von Ihrem Betriebssystem ab:

- Linux-, MacOS- oder Unix-Benutzer - `~/.aws/config`
- Windows-Benutzer - `C:\Users\USER_NAME\.aws\config`

Wenn Sie noch keine freigegebene `config`-Datei haben, können Sie diese in dem angegebenen Verzeichnis erstellen. Im folgenden Beispiel werden sowohl die Region als auch das Ausgabeformat über die `config`-Datei definiert.

```
[default]
  region=us-west-2
  output=json
```

Weitere Informationen zur Verwendung von `credentials` Dateien `config` und gemeinsam genutzten Dateien finden Sie im Referenzhandbuch für AWS SDKs und Tools unter **Gemeinsam genutzte Config- und [Credentials-Dateien](#)**.

Reihenfolge der Festlegung der Region

Im Folgenden finden Sie die Rangfolge für die Einstellung der Region:

1. Wenn eine Region an einen Client-Klassenkonstruktor übergeben wird, wird diese Region verwendet.
2. Wenn in der Umgebungsvariablen eine Region festgelegt ist, wird diese Region verwendet.
3. Andernfalls wird die in der gemeinsam genutzten Konfigurationsdatei definierte Region verwendet.

Legen Sie die Anmeldeinformationen fest

AWS verwendet Anmeldeinformationen, um festzustellen, wer Dienste anruft und ob der Zugriff auf die angeforderten Ressourcen zulässig ist.

Unabhängig davon, ob Ihr JavaScript Code in einem Webbrowser oder auf einem Node.js - Server ausgeführt wird, muss er gültige Anmeldeinformationen erhalten, bevor er über die API auf Dienste zugreifen kann. Anmeldeinformationen können pro Dienst festgelegt werden, indem Anmeldeinformationen direkt an ein Dienstobjekt übergeben werden.

Es gibt mehrere Möglichkeiten, Anmeldeinformationen festzulegen, die sich in Node.js und JavaScript in Webbrowsern unterscheiden. Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js oder Web-Browsern definieren. In jedem Fall werden die Optionen in der empfohlenen Reihenfolge dargestellt.

Bewährte Methoden für Anmeldeinformationen

Durch das ordnungsgemäße Festlegen von Anmeldeinformationen wird sichergestellt, dass Ihre Anwendung oder Ihr Browser-Skript auf die benötigten Services und Ressourcen zugreifen kann. Gleichzeitig werden Sicherheitsrisiken minimiert, die geschäftskritische Anwendungen beeinträchtigen oder vertrauliche Daten kompromittieren könnten.

Ein wichtiger Grundsatz beim Einrichten von Anmeldeinformationen lautet, immer nur die geringstmöglichen Berechtigungen zu erteilen, die für eine Aufgabe erforderlich sind. Es ist sicherer, minimale Berechtigungen für Ihre Ressourcen bereitzustellen und bei Bedarf weitere Berechtigungen hinzuzufügen, anstatt Berechtigungen bereitzustellen, die die geringsten Rechte überschreiten,

und aufgrund dessen zu einem späteren Zeitpunkt möglicherweise Sicherheitsprobleme beheben zu müssen. Wenn Sie beispielsweise nicht einzelne Ressourcen lesen und schreiben müssen, z. B. Objekte in einem Amazon S3 S3-Bucket oder einer DynamoDB-Tabelle, legen Sie für diese Berechtigungen nur Lesen fest.

Weitere Informationen zur Gewährung der geringsten Rechte finden Sie im Abschnitt [Geringste Rechte gewähren](#) des Themas Best Practices im IAM-Benutzerhandbuch.

Themen

- [Festlegen von Anmeldeinformationen in Node.js](#)
- [Anmeldeinformationen in einem Webbrowser einrichten](#)

Festlegen von Anmeldeinformationen in Node.js

Wir empfehlen, dass neue Benutzer, die lokal entwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode erhalten, einrichten AWS IAM Identity Center. Weitere Informationen finden Sie unter [SDK-Authentifizierung mit AWS](#).

Es gibt mehrere Möglichkeiten in Node.js, um dem SDK die Anmeldeinformationen bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Anwendungsentwicklung. Achten Sie beim Abrufen von Anmeldeinformationen in Node.js darauf, sich auf mehr als eine Quelle zu verlassen, z. B. auf eine Umgebungsvariable und eine JSON-Datei, die Sie laden. Sie können die Berechtigungen, unter denen Ihr Code ausgeführt wird, ändern, ohne sich dieser Änderung bewusst zu sein.

AWS SDK for JavaScript V3 stellt eine standardmäßige Anbieterkette für Anmeldeinformationen in Node.js bereit, sodass Sie einen Anbieter für Anmeldeinformationen nicht explizit angeben müssen. Die standardmäßige [Anbieterkette für Anmeldeinformationen](#) versucht, die Anmeldeinformationen aus einer Vielzahl verschiedener Quellen in einer bestimmten Priorität aufzulösen, bis eine Anmeldeinformation von einer der Quellen zurückgegeben wird. Die Anbieterkette für Anmeldeinformationen für SDK for JavaScript V3 finden Sie [hier](#).

Anbieterkette von Anmeldeinformationen

Alle -SDKs verfügen über eine Reihe von Stellen (oder Quellen), die sie überprüfen, um gültige Anmeldeinformationen für die Anforderung an einen zu erhalten AWS-Service. Nachdem gültige Anmeldeinformationen gefunden wurden, wird die Suche beendet. Diese systematische Suche wird als Standard-Anbieterkette für Anmeldeinformationen bezeichnet.

Für jeden Schritt in der Kette gibt es verschiedene Möglichkeiten, die Werte festzulegen. Das Festlegen von Werten direkt im Code hat immer Vorrang, gefolgt von der Einstellung als Umgebungsvariablen und dann in der AWS `config` freigegebenen Datei. Weitere Informationen finden Sie unter [Priorität von Einstellungen](#) im AWS Referenzhandbuch für -SDKs und Tools.

Das AWS Referenzhandbuch für SDKs und Tools enthält Informationen zu den SDK-Konfigurationseinstellungen, die von allen AWS SDKs und der verwendet werden AWS CLI. Weitere Informationen zum Konfigurieren des SDK über die AWS `config` freigegebene Datei finden Sie unter [Gemeinsam genutzte Konfigurations- und Anmeldeinformationsdateien](#). Weitere Informationen zum Konfigurieren des SDK durch Festlegen von Umgebungsvariablen finden Sie unter [Unterstützung von Umgebungsvariablen](#).

Um sich bei zu authentifizieren AWS, AWS SDK for JavaScript überprüft die die Anmeldeinformationsanbieter in der in der folgenden Tabelle aufgeführten Reihenfolge.

Methode des APIAWS SDK for JavaScript -Referenzanbieters für Anmeldeinformationen nach Priorität	Anbieter(e) verfügbarer Anmeldeinformationen	AWS Referenzhandbuch für SDKs und Tools
fromEnv()	AWS -Zugriffsschlüssel aus Umgebungsvariablen	AWS -Zugriffsschlüssel
fromSSO()	AWS IAM Identity Center. In diesem Handbuch finden Sie weitere Informationen unter SDK-Authentifizierung mit AWS .	IAM-Identity-Center-Anmeldeinformationsanbieter
fromIni()	AWS -Zugriffsschlüssel aus freigegebenen - <code>config</code> und - <code>credentials</code> Dateien	AWS -Zugriffsschlüssel
	Anbieter vertrauenswürdiger Entitäten (z. B. <code>AWS_ROLE_ARN</code>)	Übernehmen einer IAM-Rolle

Methode des APIAWS SDK for JavaScript -Referenzanbieters für Anmeldeinformationen nach Priorität	Anbieter(e) verfügbarer Anmeldeinformationen	AWS Referenzhandbuch für SDKs und Tools
	Web-Identitätstoken von AWS Security Token Service (AWS STS)	Verbinden mit Web-Identität oder OpenID Connect
	Anmeldeinformationen für Amazon Elastic Container Service (Amazon ECS)	Anbieter von Container-Anmeldeinformationen
	Amazon Elastic Compute Cloud (Amazon EC2)-Instance-Profilanmeldeinformationen (IMDS-Anmeldeinformationsanbieter)	Anbieter von IMDS-Anmeldeinformationen
	Anbieter von Prozessanmeldeinformationen	Anbieter von Prozessanmeldeinformationen
	AWS IAM Identity Center - Anmeldeinformationen	IAM-Identity-Center-Anmeldeinformationsanbieter
fromProcess()	Anbieter von Prozessanmeldeinformationen	Anbieter von Prozessanmeldeinformationen
fromTokenFile()	Web-Identitätstoken von AWS Security Token Service (AWS STS)	Verbinden mit Web-Identität oder OpenID Connect
fromContainerMetadata()	Anmeldeinformationen für Amazon Elastic Container Service (Amazon ECS)	Anbieter von Container-Anmeldeinformationen

Methode des APIAWS SDK for JavaScript -Referenzanbieters für Anmeldeinformationen nach Priorität	Anbieter(e) verfügbarer Anmeldeinformationen	AWS Referenzhandbuch für SDKs und Tools
<u>fromInstanceMetadata()</u>	Amazon Elastic Compute Cloud (Amazon EC2)-Instance-Profilanmeldeinformationen (IMDS-Anmeldeinformationsanbieter)	<u>Anbieter von IMDS-Anmeldeinformationen</u>

Wenn Sie den empfohlenen Ansatz für neue Benutzer befolgt haben, richten Sie die AWS IAM Identity Center Authentifizierung während des [SDK-Authentifizierung mit AWS](#) Themas Erste Schritte ein. Andere Authentifizierungsmethoden sind für verschiedene Situationen nützlich. Um Sicherheitsrisiken zu vermeiden, empfehlen wir, immer kurzfristige Anmeldeinformationen zu verwenden. Weitere Verfahren zur Authentifizierungsmethode finden Sie unter [Authentifizierung und Zugriff](#) im AWS Referenzhandbuch für -SDKs und Tools.

Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js laden.

Themen

- [Laden von Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2](#)
- [Laden von Anmeldeinformationen für eine Node.js-Lambda-Funktion](#)

Laden von Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2

Wenn Sie Ihre Node.js-Anwendung auf einer Amazon EC2-Instance ausführen, können Sie IAM-Rollen für Amazon EC2 nutzen, um der Instance automatisch Anmeldeinformationen bereitzustellen. Wenn Sie Ihre Instance für die Verwendung von IAM-Rollen konfigurieren, wählt das SDK automatisch die IAM-Anmeldeinformationen für Ihre Anwendung aus, sodass keine Anmeldeinformationen manuell angegeben werden müssen.

Weitere Informationen zum Hinzufügen von IAM-Rollen zu einer Amazon EC2-Instance finden Sie unter [IAM-Rollen für Amazon EC2](#).

Laden von Anmeldeinformationen für eine Node.js-Lambda-Funktion

Wenn Sie eine - AWS Lambda Funktion erstellen, müssen Sie eine spezielle IAM-Rolle erstellen, die über die Berechtigung zum Ausführen der Funktion verfügt. Diese Rolle wird Ausführungsrolle genannt. Wenn Sie eine Lambda-Funktion einrichten, müssen Sie die von Ihnen erstellte IAM-Rolle als entsprechende Ausführungsrolle angeben.

Die Ausführungsrolle stellt der Lambda-Funktion die Anmeldeinformationen bereit, die sie zum Ausführen und Aufrufen anderer Webservices benötigt. Daher müssen Sie keine Anmeldeinformationen für den Node.js-Code bereitstellen, den Sie in einer Lambda-Funktion schreiben.

Weitere Informationen zum Erstellen einer Lambda-Ausführungsrolle finden Sie unter [Verwalten von Berechtigungen: Verwenden einer IAM-Rolle \(Ausführungsrolle\)](#) im AWS Lambda - Entwicklerhandbuch.

Anmeldeinformationen in einem Webbrowser einrichten

Es gibt mehrere Möglichkeiten, um dem SDK die Anmeldeinformationen aus Browser-Skripts bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Entwicklung eines Skripts.

So können Sie Ihre Anmeldeinformationen in der Reihenfolge ihrer Empfehlungen angeben:

1. Verwenden von Amazon Cognito Identity zur Benutzerauthentifizierung und Bereitstellung von Anmeldeinformationen
2. Verwenden von Web-Verbundidentitäten

Warning

Wir empfehlen nicht, Ihre AWS Anmeldeinformationen in Ihren Skripten fest zu codieren. Die Hartcodierung der Anmeldeinformationen setzt die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel dem Risiko einer Offenlegung aus.

Themen

- [Verwenden Sie Amazon Cognito Identity, um Benutzer zu authentifizieren](#)

Verwenden Sie Amazon Cognito Identity, um Benutzer zu authentifizieren

Die empfohlene Methode zum Abrufen von AWS Anmeldeinformationen für Ihre Browserskripte ist die Verwendung des Amazon Cognito Identity-Anmeldeinformationsclients `CognitoIdentityClient`. Amazon Cognito ermöglicht die Authentifizierung von Benutzern über externe Identitätsanbieter.

Um Amazon Cognito Identity verwenden zu können, müssen Sie zunächst einen Identitätspool in der Amazon Cognito Cognito-Konsole erstellen. Ein Identitäten-Pool stellt die Gruppe von Identitäten dar, die Ihre Anwendung für Ihre Benutzer bereitstellt. Die den Benutzern zugewiesenen Identitäten identifizieren jedes Benutzerkonto eindeutig. Amazon-Cognito-Identitäten sind keine Anmeldeinformationen. Sie werden mithilfe der Unterstützung von Web Identity Federation in AWS Security Token Service (AWS STS) gegen Anmeldeinformationen ausgetauscht.

Amazon Cognito unterstützt Sie bei der Verwaltung der Abstraktion von Identitäten über mehrere Identitätsanbieter hinweg. Die geladene Identität wird dann gegen Anmeldeinformationen in AWS STS ausgetauscht.

Konfigurieren Sie das Amazon Cognito Identity-Anmeldeinformationsobjekt

Wenn Sie noch keinen erstellt haben, erstellen Sie einen Identitätspool, den Sie mit Ihren Browserskripten in der [Amazon Cognito Cognito-Konsole](#) verwenden können, bevor Sie Ihren Amazon Cognito Cognito-Client konfigurieren. Erstellen Sie sowohl authentifizierte als auch nicht authentifizierte IAM-Rollen für Ihren Identitätspool und ordnen Sie sie zu. Weitere Informationen finden Sie unter [Tutorial: Erstellen eines Identitätspools](#) im Amazon Cognito Developer Guide.

Bei nicht authentifizierten Benutzern wurde die Identität nicht verifiziert. Daher eignet sich diese Rolle für Gastbenutzer Ihrer App oder für Fälle, in denen es egal ist, ob die Identität der Benutzer verifiziert wurde. Authentifizierte Benutzer melden sich bei Ihrer Anwendung über einen Drittanbieter an, der ihre Identität überprüft. Vergewissern Sie sich, dass Sie die Berechtigungen der Ressourcen entsprechend anpassen, damit Sie keinen Zugriff von nicht authentifizierten Benutzern darauf gewähren.

Nachdem Sie einen Identitätspool konfiguriert haben, verwenden Sie die `fromCognitoIdentityPool` Methode von, `@aws-sdk/credential-providers` um die Anmeldeinformationen aus dem Identitätspool abzurufen. Im folgenden Beispiel für die Erstellung eines Amazon S3 S3-Clients ersetzen Sie *AWS_REGION* durch die *Region* und *IDENTITY_POOL_ID* durch die *Identitätspool-ID*.

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
      // Optional tokens, used for authenticated login.
    },
  })
});
```

Die optionale `logins`-Eigenschaft ist eine Abbildung der Namen des Identitätsanbieters auf die Identitäts-Token für diese Anbieter. Wie Sie den Token von Ihrem Identitätsanbieter erhalten, hängt davon ab, welchen Anbieter Sie verwenden. Wenn Sie beispielsweise einen Amazon Cognito Cognito-Benutzerpool als Authentifizierungsanbieter verwenden, könnten Sie eine Methode verwenden, die der folgenden ähnelt.

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
let loginData = {
  [COGNITO_ID]: idToken,
};
const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
```

```
var idtoken = window.location.href;
var idtoken1 = idtoken.split("=")[1];
var idtoken2 = idtoken1.split("&")[0];
var idtoken3 = idtoken2.split("&")[0];
return idtoken3;
};
```

Wechseln Sie zwischen nicht authentifizierten Benutzern und authentifizierten Benutzern

Amazon Cognito unterstützt sowohl authentifizierte als auch nicht authentifizierte Benutzer. Nicht authentifizierte Benutzer erhalten Zugriff auf Ihre Ressourcen, auch wenn sie nicht über Ihre Identitätsanbieter angemeldet sind. Dieser Grad des Zugriffs ist nützlich, um Inhalte für Benutzer anzuzeigen, bevor diese sich anmelden. Jeder nicht authentifizierte Benutzer hat eine eindeutige Identität in Amazon Cognito, obwohl er nicht einzeln angemeldet und authentifiziert wurde.

Anfänglich nicht authentifizierter Benutzer

Benutzer beginnen in der Regel mit der nicht authentifizierten Rolle, für die Sie die Eigenschaft für die Anmeldeinformationen Ihres Konfigurationsobjekts ohne eine `logins`-Eigenschaft festlegen. In diesem Fall könnten Ihre Standardanmeldedaten wie folgt aussehen:

```
// Import the required AWS SDK for JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = fromCognitoIdentityPool({
  identityPoolId: 'IDENTITY_POOL_ID',
  clientConfig: { region: REGION } // Configure the underlying CognitoIdentityClient.
});
```

Wechseln Sie zum authentifizierten Benutzer

Wenn sich ein nicht authentifizierter Benutzer bei einem Identitätsanbieter anmeldet und Sie über ein Token verfügen, können Sie den Benutzer von nicht authentifiziert auf authentifiziert umstellen, indem Sie eine benutzerdefinierte Funktion aufrufen, die das Anmeldeinformationsobjekt aktualisiert und das Token hinzufügt. `logins`

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;
```

```
// Expire credentials to refresh them on the next request
creds.expired = true;
}
```

Überlegungen zu Node.js

Der Code Node.js ist es zwar JavaScript, aber die Verwendung von AWS SDK for JavaScript in Node.js kann sich von der Verwendung des SDK in Browserskripts unterscheiden. Einige API-Methoden funktionieren in Node.js, jedoch nicht in Browser-Skripts und umgekehrt. Die erfolgreiche Verwendung einiger APIs hängt davon ab, wie vertraut Sie mit häufigen Node.js-Codierungsmustern sind, beispielsweise dem Importieren und Verwenden anderer Node.js-Module wie des File System (fs)-Moduls.

Verwenden Sie die integrierten Node.js -Module

Node.js bietet eine Reihe von integrierten Modulen, die Sie verwenden können ohne sie installieren zu müssen. Um diese Module verwenden zu können, müssen Sie ein Objekt mit der `require`-Methode erstellen und den Modulnamen angeben. Wenn beispielsweise das integrierte HTTP-Modul enthalten sein soll, geben Sie Folgendes ein.

```
import http from 'http';
```

Rufen Sie Methoden des Moduls ab, als würde es sich um Methoden dieses Objekts handeln. Das folgende Beispiel zeigt Code, der eine HTML-Datei liest.

```
// include File System module
import fs from "fs";
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
    throw err;
  } else {
    // Successful file read
  }
});
```

Eine vollständige Liste aller integrierten Module, die Node.js bereitstellt, finden Sie in der [Node.js - Dokumentation](#) auf der Website Node.js.

Verwenden Sie npm-Pakete

Zusätzlich zu den integrierten Modulen können Sie auch Code von Drittanbietern aus dem npm Paketmanager Node.js einbinden und integrieren. Hierbei handelt es sich um ein Repository mit Open-Source-Node.js-Paketen sowie um eine Befehlszeilen-Schnittstelle für die Installation dieser Pakete. Weitere Informationen npm und eine Liste der derzeit verfügbaren Pakete finden Sie unter <https://www.npmjs.com>. Weitere Informationen zu weiteren Node.js Paketen, die Sie verwenden können, finden Sie [hier auf GitHub](#).

MaxSockets in Node.js konfigurieren

In Node.js können Sie die maximale Anzahl der Verbindungen pro Ursprungsserver festlegen. Wenn `maxSockets` festgelegt wurde, setzt der Low-Level-HTTP-Client die Anforderungen auf eine Warteliste und ordnet sie Sockets zu, sobald diese verfügbar werden.

Mit dieser Option können Sie eine Obergrenze für die Anzahl der gleichzeitigen Anforderungen an einen bestimmten Ursprungsserver angeben. Indem Sie diesen Wert senken, kann die Anzahl der maximalen Ablehnungen oder Timeout-Fehler reduziert werden. Es kann jedoch auch die Speichernutzung erhöhen, da Anforderungen so lange in eine Warteschlange gesetzt werden, bis ein Socket verfügbar wird.

Das folgende Beispiel zeigt, wie die Einstellungen `maxSockets` für einen DynamoDB-Client vorgenommen werden.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import https from "https";
let agent = new https.Agent({
  maxSockets: 25
});

let dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    requestTimeout: 3_000,
    httpsAgent: agent
  });
});
```

Das SDK für JavaScript verwendet den `maxSockets` Wert 50, wenn Sie keinen Wert oder kein Objekt `Agent` angeben. Wenn Sie ein `Agent` Objekt angeben, wird sein `maxSockets` Wert

verwendet. Weitere Informationen zur Einstellung `maxSockets` in Node.js finden Sie in der [Dokumentation zu Node.js](#).

Ab Version 3.521.0 von können Sie AWS SDK for JavaScript die folgende [Kurzsyntax](#) zur Konfiguration verwenden. `requestHandler`

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  requestHandler: {
    requestTimeout: 3_000,
    httpsAgent: { maxSockets: 25 },
  },
});
```

Verbindungen mit Keep-Alive in Node.js wiederverwenden

Der standardmäßige Node.js-HTTP/HTTPS-Agent erstellt eine neue TCP-Verbindung für jede neue Anforderung. Um die Kosten für den Aufbau einer neuen Verbindung zu vermeiden, werden TCP-Verbindungen AWS SDK for JavaScript standardmäßig wiederverwendet.

Bei kurzlebigen Vorgängen, wie z. B. Amazon DynamoDB DynamoDB-Abfragen, kann der Latenzaufwand beim Einrichten einer TCP-Verbindung größer sein als der Vorgang selbst. Da die [DynamoDB-Verschlüsselung im Ruhezustand](#) integriert ist [AWS KMS](#), kann es außerdem zu Latenzen kommen, weil die Datenbank für jeden Vorgang neue AWS KMS Cacheeinträge einrichten muss.

Wenn Sie TCP-Verbindungen nicht wiederverwenden möchten, können Sie die Wiederverwendung dieser Verbindungen `keepAlive live with` für jeden einzelnen Service deaktivieren, wie im folgenden Beispiel für einen DynamoDB-Client gezeigt.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";

const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpsAgent: new Agent({ keepAlive: false })
  })
});
```

Wenn `keepAlive` aktiviert, können Sie auch die Anfangsverzögerung für TCP-Keep-Alive-Pakete mit `keepAliveMsecs` festlegen, die standardmäßig 1000 ms beträgt. Weitere Informationen finden Sie in der [Node.js-Dokumentation](#).

Konfigurieren Sie Proxys für Node.js

Wenn Sie keine direkte Verbindung zum Internet herstellen können, unterstützt das SDK für die Verwendung von HTTP- oder HTTPS-Proxys über einen HTTP-Agenten eines Drittanbieters.

[Um einen HTTP-Agenten eines Drittanbieters zu finden, suchen Sie bei npm nach „HTTP-Proxy“.](#)

Um einen HTTP-Agent-Proxy eines Drittanbieters zu installieren, geben Sie an der Befehlszeile Folgendes ein, wobei *PROXY* der Name des npm Pakets ist.

```
npm install PROXY --save
```

Um einen Proxy in Ihrer Anwendung zu verwenden, verwenden Sie die `httpsAgent` Eigenschaft `httpAgent` and, wie im folgenden Beispiel für einen DynamoDB-Client gezeigt.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from '@smithy/node-http-handler';
import { HttpsProxyAgent } from 'hpagent';
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  }),
});
```

Note

`httpAgent` ist nicht dasselbe wie `httpsAgent`, und da die meisten Aufrufe vom Client an `https` erfolgen, sollten beide gesetzt werden.

Registrieren Sie Zertifikatspakete in Node.js

Die standardmäßigen Vertrauensspeicher für Node.js enthalten die Zertifikate, die für den Zugriff auf AWS Dienste erforderlich sind. In einigen Fällen könnte es vorteilhaft sein, nur eine bestimmte Gruppe von Zertifikaten einzuschließen.

In diesem Beispiel wird ein bestimmtes Zertifikat auf der Festplatte verwendet, mit dem ein `https.Agent` erstellt wird, der alle Verbindungen ablehnt, die nicht über das vorgesehene Zertifikat verfügen. Das neu erstellte Objekt `https.Agent` wird dann vom DynamoDB-Client verwendet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Überlegungen zum Browser-Skript

In den folgenden Themen werden besondere Überlegungen zur Verwendung von AWS SDK for JavaScript In-Browser-Skripts beschrieben.

Themen

- [Erstellen Sie das SDK für Browser](#)
- [Cross-Origin Resource Sharing \(CORS\)](#)
- [Anwendungen mit Webpack bündeln](#)

Erstellen Sie das SDK für Browser

Im Gegensatz zum SDK für JavaScript Version 2 (V2) wird V3 nicht als JavaScript Datei mit Unterstützung für eine Reihe von Standarddiensten bereitgestellt. Stattdessen können Sie mit V3 nur das SDK für die benötigten JavaScript Dateien bündeln und in den Browser aufnehmen, wodurch der Overhead reduziert wird. Wir empfehlen, Webpack zu verwenden, um das erforderliche SDK für JavaScript Dateien und alle zusätzlichen Pakete von Drittanbietern, die Sie benötigen, in einer einzigen Javascript Datei zu bündeln und diese mithilfe eines `<script>` Tags in Browserskripte zu laden. Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#). Ein Beispiel, das Webpack verwendet, um das V3-SDK JavaScript in einen Browser zu laden, finden Sie unter [Erstellen Sie eine App zum Senden von Daten an DynamoDB](#).

Wenn Sie mit dem SDK außerhalb einer Umgebung arbeiten, in der CORS in Ihrem Browser durchgesetzt wird, und wenn Sie Zugriff auf alle vom SDK bereitgestellten Dienste für haben möchten JavaScript, können Sie lokal eine benutzerdefinierte Kopie des SDK erstellen, indem Sie das Repository klonen und dieselben Build-Tools ausführen, mit denen die gehostete Standardversion des SDK erstellt wird. In den folgenden Abschnitten werden die Schritte erklärt, anhand derer das SDK zusammen mit zusätzlichen Services und API-Versionen erstellt werden kann.

Verwenden Sie den SDK Builder, um das SDK für zu erstellen JavaScript

Note

Amazon Web Services Version 3 (V3) unterstützt Browser Builder nicht mehr. Um die Bandbreitennutzung von Browseranwendungen zu minimieren, empfehlen wir Ihnen, benannte Module zu importieren und sie zu bündeln, um die Größe zu reduzieren. Weitere Informationen zur Bündelung finden Sie unter [Anwendungen mit Webpack bündeln](#).

Cross-Origin Resource Sharing (CORS)

Bei Cross-Origin Resource Sharing oder CORS handelt es sich um eine Sicherheitsfunktion der modernen Webbrowser. Es ermöglicht Webbrowsern zu verhandeln, welche Domänen Anforderungen von externen Websites oder Services senden können.

CORS ist ein wichtiger Aspekt bei der Entwicklung von Browseranwendungen mit dem AWS SDK for JavaScript , da die meisten Anforderungen an Ressourcen an eine externe Domäne, wie z. B. der Endpunkt für einen Webservice, gesendet werden. Wenn Ihre JavaScript Umgebung CORS-Sicherheit erzwingt, müssen Sie CORS mit dem Service konfigurieren.

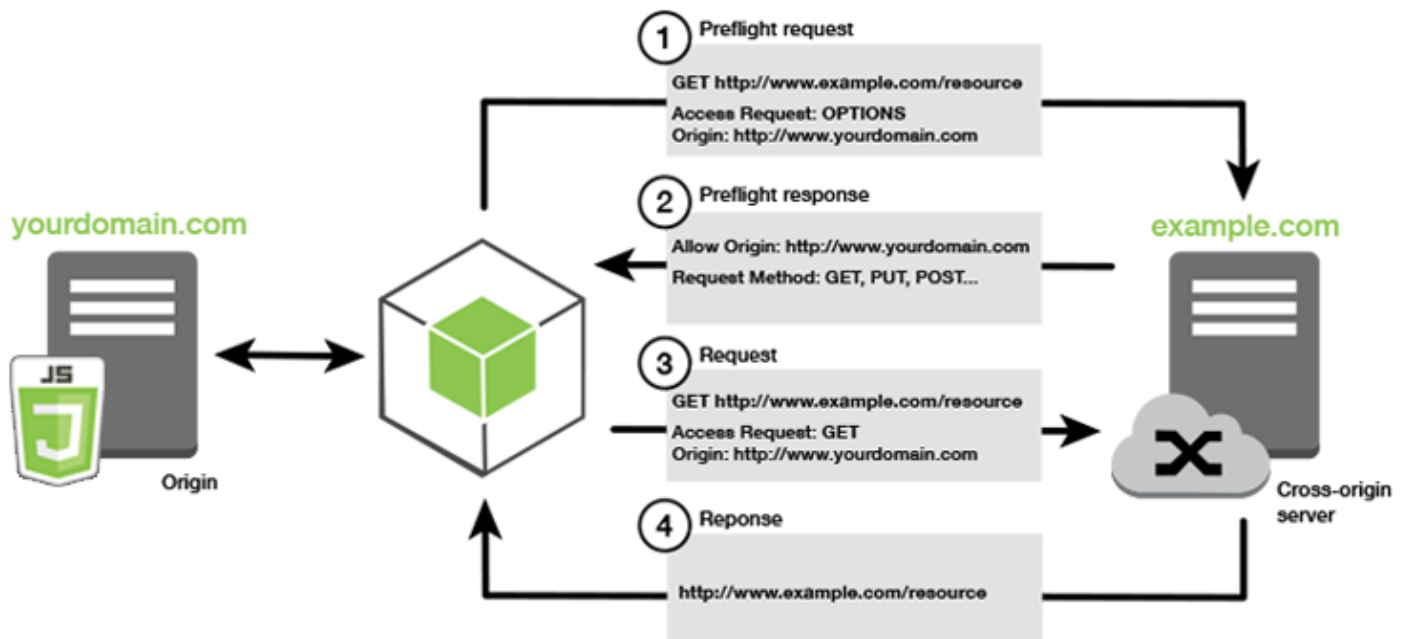
CORS bestimmt anhand der folgenden Faktoren, ob die gemeinsame Nutzung von Ressourcen in einer ursprungsübergreifenden Anforderung zugelassen werden soll:

- Der spezifischen Domäne, die die Anforderung sendet
- Dem Typ der HTTP-Anforderung, die gesendet wird (GET, PUT, POST, DELETE usw.)

Funktionsweise von CORS

Im einfachsten Fall sendet Ihr Browser-Skript eine GET-Anforderung für eine Ressource von einem Server in einer anderen Domäne. Abhängig von der CORS-Konfiguration dieses Servers, wenn die Anforderung von einer Domäne stammt, die berechtigt ist, GET-Anforderungen abzusenden, reagiert der ursprungsübergreifende Server, indem er die angeforderte Ressource zurück gibt.

Wenn entweder die anfordernde Domäne oder der Typ der HTTP-Anforderung nicht autorisiert ist, wird die Anforderung abgelehnt. CORS jedoch ermöglicht ein Preflight der Anforderung, bevor sie tatsächlich abgesendet wird. In diesem Fall wird eine Preflight-Anforderung ausgeführt, in der die OPTIONS-Zugriffsanforderungsoperation gesendet wird. Wenn die CORS-Konfiguration des ursprungsübergreifenden Servers Zugriff auf die anfordernde Domäne gewährt, sendet der Server eine Preflight-Antwort zurück, die alle HTTP-Anforderungstypen aufführt, die die anfordernde Domäne für die angeforderte Ressource vornehmen kann.



Ist eine CORS-Konfiguration erforderlich?

Amazon S3-Buckets erfordern eine CORS-Konfiguration, bevor Sie Vorgänge für sie ausführen können. In einigen JavaScript Umgebungen wird CORS möglicherweise nicht erzwungen, daher ist die Konfiguration von CORS nicht erforderlich. Wenn Sie Ihre Anwendung beispielsweise aus einem Amazon S3-Bucket hosten und von `*.s3.amazonaws.com` oder einem anderen spezifischen Endpunkt auf Ressourcen zugreifen, greifen Ihre Anforderungen nicht auf eine externe Domain zu. Aus diesem Grund benötigt diese Konfiguration kein CORS. In diesem Fall wird CORS weiterhin für andere Services als Amazon S3 verwendet.

Konfigurieren von CORS für einen Amazon S3-Bucket

Sie können einen Amazon S3-Bucket für die Verwendung von CORS in der Amazon S3-Konsole konfigurieren.

Wenn Sie CORS in der AWS Web Services Management Console konfigurieren, müssen Sie JSON verwenden, um eine CORS-Konfiguration zu erstellen. Die neue AWS Web Services Management Console unterstützt nur JSON CORS-Konfigurationen.

Important

In der neuen AWS Web Services Management Console muss die CORS-Konfiguration JSON sein.

1. Öffnen Sie in der AWS Web Services Management Console die Amazon S3-Konsole, suchen Sie den Bucket, den Sie konfigurieren möchten, und aktivieren Sie das entsprechende Kontrollkästchen.
2. Wählen Sie im sich öffnenden Bereich Berechtigungen aus.
3. Wählen Sie auf der Registerkarte Berechtigung die Option CORS-Konfiguration aus.
4. Geben Sie Ihre CORS-Konfiguration im CORS-Konfigurationseditor ein und wählen Sie dann Speichern aus.

Eine CORS-Konfiguration ist eine XML-Datei mit einer Reihe von Regeln innerhalb eines `<CORSRule>`. Eine Konfiguration kann bis zu 100 Regeln enthalten. Eine Regel wird durch eines der folgenden Tags definiert:

- `<AllowedOrigin>` – Gibt Domänenursprünge an, die Sie zulassen, um domänenübergreifende Anforderungen zu stellen.
- `<AllowedMethod>` – Gibt einen Anforderungstyp an, den Sie in domänenübergreifenden Anforderungen zulassen (GET, PUT, POST, DELETE, HEAD).
- `<AllowedHeader>` – Gibt die Header an, die in einer Preflight-Anforderung zulässig sind.

Beispielkonfigurationen finden [Sie unter Wie konfiguriere ich CORS für meinen Bucket?](#) im Benutzerhandbuch für Amazon Simple Storage Service.

CORS-Konfigurationsbeispiel

Das folgende CORS-Konfigurationsbeispiel ermöglicht es einem Benutzer, Objekte innerhalb eines Buckets aus der Domain anzuzeigen, hinzuzufügen, zu entfernen oder zu aktualisieren `example.org`. Wir empfehlen Ihnen jedoch, die `<AllowedOrigin>` auf die Domain Ihrer Website zu beschränken. Sie können "*" angeben, sodass jeder Ursprung zulässig ist.

Important

In der neuen S3-Konsole muss die CORS-Konfiguration JSON sein.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "HEAD",
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "https://www.example.org"
    ],
    "ExposeHeaders": [
      "ETag",
      "x-amz-meta-custom-header"
    ]
  }
]
```

Diese Konfiguration autorisiert den Benutzer nicht, Aktionen für den Bucket auszuführen. Es ermöglicht dem Sicherheitsmodell des Browsers, eine Anforderung an Amazon S3 zuzulassen. Berechtigungen müssen über Bucket-Berechtigungen oder IAM-Rollenberechtigungen konfiguriert werden.

Sie können verwenden `ExposeHeader`, damit das SDK Antwort-Header liest, die von Amazon S3 zurückgegeben werden. Lesen Sie beispielsweise den `-ETagHeader` aus einem PUT oder mehrteiligen Upload. Sie müssen das `-ExposeHeaderTag` in Ihre Konfiguration aufnehmen, wie im vorherigen Beispiel gezeigt. Das SDK kann nur auf Header zugreifen, die über die CORS-Konfiguration bereitgestellt werden. Wenn Sie Metadaten für das Objekt festlegen, werden Werte als Header mit dem Präfix `x-amz-meta-` zurückgegeben, wie z. B. `x-amz-meta-my-custom-header`, und müssen auch auf die gleiche Weise bereitgestellt werden.

Anwendungen mit Webpack bündeln

Die Verwendung von Codemodulen durch Webanwendungen in Browserskripten oder Node.js erzeugt Abhängigkeiten. Diese Codemodule können eigene Abhängigkeiten haben, was zu

einer Sammlung von miteinander verbundenen Modulen führt, die Ihre Anwendung benötigt, um zu funktionieren. Um Abhängigkeiten zu verwalten, können Sie einen Modul-Bundler wie `webpack` verwenden.

Der `webpack` Modul-Bundler analysiert Ihren Anwendungscode und sucht nach `import` oder `require` Anweisungen, um Pakete zu erstellen, die alle Komponenten enthalten, die Ihre Anwendung benötigt. Auf diese Weise können die Komponenten einfach über eine Webseite bereitgestellt werden. Das SDK für JavaScript kann in `webpack` als eine der Abhängigkeiten enthalten sein, die im Ausgabe-Bundle enthalten sein sollen.

Weitere Informationen zu `webpack` finden Sie im [Webpack-Modul-Bundler](#) auf GitHub.

Installieren von Webpack

Um den `webpack` Modul-Bundler zu installieren, müssen Sie zuerst `npm`, den Node.js-Paketmanager, installiert haben. Geben Sie den folgenden Befehl ein, um die `webpack` CLI und das JavaScript Modul zu installieren.

```
npm install --save-dev webpack
```

Um das `path` Modul für die Arbeit mit Datei- und Verzeichnispfaden zu verwenden, das automatisch mit `Webpack` installiert wird, müssen Sie möglicherweise das `Node.js-path-browserify` Paket installieren.

```
npm install --save-dev path-browserify
```

Webpack konfigurieren

Standardmäßig sucht `Webpack` nach einer JavaScript Datei mit dem Namen `webpack.config.js` im Stammverzeichnis Ihres Projekts. Diese Datei enthält Ihre Konfigurationsoptionen. Im Folgenden finden Sie ein Beispiel für eine `webpack.config.js` Konfigurationsdatei für `Webpack` Version 5.0.0 und höher.

Note

Die `Webpack`-Konfigurationsanforderungen variieren je nach installierter `Webpack`-Version. Weitere Informationen finden Sie in der [Webpack-Dokumentation](#).

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js)],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Enable WebPack to use the 'path' package.
  resolve:{
  fallback: { path: require.resolve("path-browserify")}
}
/**
 * In Webpack version v2.0.0 and earlier, you must tell
 * webpack how to use "json-loader" to load 'json' files.
 * To do this Enter 'npm --save-dev install json-loader' at the
 * command line to install the "json-loader" package, and include the
 * following entry in your webpack.config.js.
 * module: {
   rules: [{test: /\.json$/, use: use: "json-loader"}]
 }
 **/
};
```

In diesem Beispiel `browser.js` wird als Eintrittspunkt angegeben. Der Einstiegspunkt ist die Datei, die webpack verwendet, um nach importierten Modulen zu suchen. Der Name für die Ausgabedatei ist als `bundle.js` angegeben. Diese Ausgabedatei enthält alle , die JavaScript die Anwendung zum Ausführen benötigt. Wenn der im Einstiegspunkt angegebene Code andere Module importiert oder erfordert, z. B. das SDK für JavaScript, wird dieser Code gebündelt, ohne ihn in der Konfiguration angeben zu müssen.

Webpack ausführen

Um eine Anwendung für die Verwendung von zu erstellenwebpack, fügen Sie dem `-scripts`Objekt in Ihrer `-package.json`Datei Folgendes hinzu.

```
"build": "webpack"
```


Im Folgenden finden Sie eine `package.json` Beispieldatei, die das Hinzufügen von `demonstriertwebpack`.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

Geben Sie den folgenden Befehl ein, um Ihre Anwendung zu erstellen.

```
npm run build
```

Der `webpack` Modul-Bundler generiert dann die JavaScript Datei, die Sie im Stammverzeichnis Ihres Projekts angegeben haben.

Verwenden des Webpack-Pakets

Um das Paket in einem Browserskript zu verwenden, können Sie das Paket mithilfe eines `<script>` Tags integrieren, wie im folgenden Beispiel gezeigt.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Amazon SDK with webpack</title>
  </head>
  <body>
    <div id="list"></div>
```

```
    <script src="bundle.js"></script>
  </body>
</html>
```

Bündel für Node.js

Sie können verwenden, webpack um Pakete zu generieren, die in Node.js ausgeführt werden, indem Sie node als Ziel in der Konfiguration angeben.

```
target: "node"
```

Dies ist nützlich, wenn eine Node.js-Anwendung in einer Umgebung ausgeführt wird, in der Speicherplatz begrenzt ist. Hier sehen Sie ein Beispiel für eine `webpack.config.js`-Konfiguration mit der Angabe von Node.js als Ausgabeziel.

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
  target: "node",
  // Enable WebPack to use the 'path' package.
  resolve:{
  fallback: { path: require.resolve("path-browserify")}
  /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader" package, and include the
   * following entry in your webpack.config.js.
   */
  module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
  **/
};
```

Arbeiten Sie mit AWS Diensten im SDK für JavaScript

Die AWS SDK for JavaScript Version 3 bietet Zugriff auf Dienste, die sie über eine Sammlung von Client-Klassen unterstützt. Von diesen Client-Klassen erstellen Sie Service-Schnittstellenobjekte, die allgemein als Service-Objekte bezeichnet werden. Jeder unterstützte AWS Dienst hat eine oder mehrere Clientklassen, die Low-Level-APIs für die Nutzung von Servicefunktionen und Ressourcen bieten. Beispielsweise sind Amazon DynamoDB DynamoDB-APIs über die DynamoDB Klasse verfügbar.

Die über das SDK bereitgestellten Dienste JavaScript folgen dem Anfrage-Antwort-Muster, um Nachrichten mit aufrufenden Anwendungen auszutauschen. In diesem Muster sendet der Code, der einen Service aufruft, eine HTTP-/HTTPS-Anforderung an einen Endpunkt für den Service. Die Anforderung enthält die erforderlichen Parameter für einen erfolgreichen Aufruf der spezifischen Funktion. Der Service, der aufgerufen wird, generiert eine Antwort, die an den Anforderer zurückgesendet wird. Die Antwort enthält Daten, wenn die Operation erfolgreich war, oder Fehlerinformationen, wenn sie nicht erfolgreich war.

Das Aufrufen eines AWS Dienstes umfasst den gesamten Anforderungs- und Antwortzyklus eines Vorgangs an einem Dienstobjekt, einschließlich aller Wiederholungsversuche. Eine Anfrage enthält null oder mehr Eigenschaften als JSON-Parameter. Die Antwort ist in einem Objekt gekapselt, das sich auf den Vorgang bezieht, und wird über eine von mehreren Techniken, z. B. eine Rückruffunktion oder eine Zusage, an den Anforderer zurückgegeben. JavaScript

Themen

- [Serviceobjekte erstellen und aufrufen](#)
- [Rufen Sie Dienste asynchron auf](#)
- [Erstellen Sie Service-Client-Anfragen](#)
- [Bearbeiten Sie die Antworten der Servicekunden](#)
- [Arbeiten Sie mit JSON](#)
- [SDK für JavaScript Codebeispiele](#)

Serviceobjekte erstellen und aufrufen

Die JavaScript API unterstützt die meisten verfügbaren AWS Dienste. Jeder Dienst in der JavaScript API stellt eine Clientklasse mit einer send Methode bereit, mit der Sie jede API aufrufen, die der

Dienst unterstützt. Weitere Informationen zu Serviceklassen, Vorgängen und Parametern in der JavaScript API finden Sie in der [API-Referenz](#).

Wenn Sie das SDK in Node.js verwenden, fügen Sie Ihrer Anwendung das SDK-Paket für jeden Dienst hinzuimport, den Sie benötigen. Dadurch werden alle aktuellen Dienste unterstützt. Im folgenden Beispiel wird ein Amazon S3 S3-Serviceobjekt in der us-west-1 Region erstellt.

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
const s3Client = new S3Client({
  region: "us-west-1"
});
```

Geben Sie die Parameter des Serviceobjekts an

Wenn Sie eine Methode eines Service-Objekts aufrufen, übergeben Sie die JSON-Parameter wie für die API erforderlich. Um beispielsweise in Amazon S3 ein Objekt für einen bestimmten Bucket und Schlüssel abzurufen, übergeben Sie der `GetObjectCommand` Methode die folgenden Parameter von `S3Client`. Weitere Informationen zum Übergeben von JSON-Parametern finden Sie unter [Arbeiten Sie mit JSON](#).

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

Weitere Informationen zu Amazon S3 S3-Parametern finden Sie unter [@aws-sdk/client-s3](#) in der API-Referenz.

Rufen Sie Dienste asynchron auf

Alle über das SDK ausgeführten Anforderungen sind asynchron. Dies ist wichtig, wenn Sie Browserskripts schreiben. JavaScript Die Ausführung in einem Webbrowser hat normalerweise nur einen einzigen Ausführungsthread. Nach einem asynchronen Aufruf eines AWS Dienstes wird das Browserskript weiter ausgeführt und kann dabei versuchen, Code auszuführen, der von diesem asynchronen Ergebnis abhängt, bevor er zurückkehrt.

Zu asynchronen Aufrufen eines AWS Dienstes gehört auch die Verwaltung dieser Aufrufe, sodass Ihr Code nicht versucht, Daten zu verwenden, bevor die Daten verfügbar sind. Die Themen in diesem Abschnitt erklären, wie wichtig es ist, asynchrone Aufrufe zu verwalten, und erläutern die verschiedenen Verwaltungstechniken im Detail.

Sie können zwar jede dieser Techniken verwenden, um asynchrone Aufrufe zu verwalten, wir empfehlen jedoch, `async/await` für allen neuen Code zu verwenden.

async/await

Wir empfehlen Ihnen, diese Technik zu verwenden, da sie das Standardverhalten in Version 3 ist. `versprechen`

Verwenden Sie diese Technik in Browsern, die `Async/Await` nicht unterstützen.

Rückruf

Vermeiden Sie Rückrufe, außer in sehr einfachen Fällen. Sie könnten es jedoch für Migrationsszenarien nützlich finden.

Themen

- [Asynchrone Aufrufe verwalten](#)
- [Verwenden Sie `async/await`](#)
- [JavaScript Verwenden Sie Versprechen](#)
- [Verwenden Sie eine anonyme Rückruffunktion](#)

Asynchrone Aufrufe verwalten

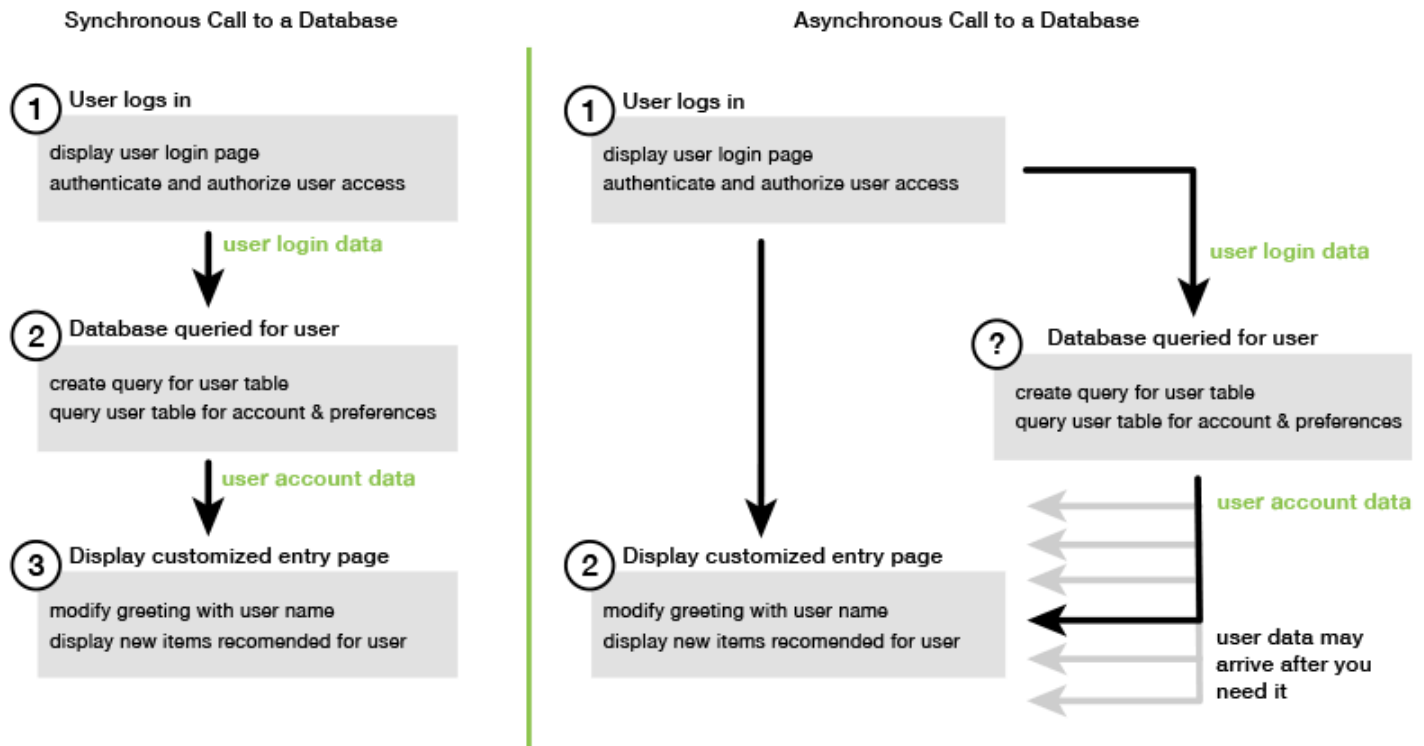
Beispiel: Die Startseite einer E-Commerce-Website bietet wiederkehrenden Kunden die Möglichkeit, sich anzumelden. Die Kunden, die sich anmelden, profitieren u. a. davon, dass sich die Website nach der Anmeldung an ihre besonderen Präferenzen anpasst. Dafür ist Folgendes erforderlich:

1. Der Kunde muss sich anmelden und mit seinen Anmeldedaten validiert werden.
2. Die Präferenzen des Kunden werden von einer Kundendatenbank angefordert.
3. Die Datenbank stellt die Präferenzen des Kunden bereit, mit denen die Website angepasst wird, bevor die Seite geladen wird.

Wenn diese Aufgaben synchron ausgeführt werden, muss jede abgeschlossen sein, bevor die nächste starten kann. Das Laden der Webseite kann erst abgeschlossen werden, wenn die Kundeneinstellungen aus der Datenbank zurückgegeben werden. Nachdem die Datenbankabfrage an den Server gesendet wurde, kann der Empfang der Kundendaten jedoch aufgrund von

Netzwerkengpässen, besonders hohem Datenverkehr oder einer schlechten Verbindung eines Mobilgeräts sich verzögern oder sogar fehlschlagen.

Um zu verhindern, dass die Website unter diesen Bedingungen einfriert, rufen Sie die Datenbank asynchron auf. Nachdem der Datenbankaufruf ausgeführt und Ihre asynchrone Anforderung gesendet wurde, wird Ihr Code wie erwartet weiter ausgeführt. Wenn Sie die Antwort eines asynchronen Aufrufs nicht ordnungsgemäß verwalten, versucht Ihr Code möglicherweise im Fall, dass diese Daten noch nicht verfügbar sind, Informationen zu verwenden, die er von der Datenbank zurückerwartet.



Verwenden Sie `async/await`

Anstatt Versprechen zu verwenden, sollten Sie `async/await` verwenden. Async-Funktionen sind einfacher und benötigen weniger Boilerplate als die Verwendung von Versprechen. `await` kann nur in einer asynchronen Funktion verwendet werden, um asynchron auf einen Wert zu warten.

Das folgende Beispiel verwendet `async/await`, um all Ihre Amazon DynamoDB-Tabellen in aufzulisten. `us-west-2`

i Note

Damit dieses Beispiel ausgeführt werden kann:

- Installieren Sie den AWS SDK for JavaScript DynamoDB-Client, indem Sie ihn npm `install @aws-sdk/client-dynamodb` in die Befehlszeile Ihres Projekts eingeben.
- Stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen korrekt konfiguriert haben. Weitere Informationen finden Sie unter [Legen Sie die Anmeldeinformationen fest](#).

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
(async function () {
  const dbClient = new DynamoDBClient({ region: "us-west-2" });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err)
  }
})();
```

Note

Nicht alle Browser unterstützen Async/Await. Eine Liste von Browsern mit [Async/Await-Unterstützung](#) finden Sie unter [Async-Funktionen](#).

JavaScript Verwenden Sie Versprechen

Verwenden Sie die AWS SDK for JavaScript v3-Methode (`ListTablesCommand`) des Service-Clients, um den Service aufzurufen und den asynchronen Ablauf zu verwalten, anstatt Callbacks zu verwenden. Das folgende Beispiel zeigt, wie Sie die Namen Ihrer Amazon DynamoDB-Tabellen abrufen können. `us-west-2`

```
import {
  DynamoDBClient,
  ListTablesCommand
```

```
} from "@aws-sdk/client-dynamodb";
const dbClient = new DynamoDBClient({ region: 'us-west-2' });

dbClient.listTables(new ListTablesCommand({}))
  .then(response => {
    console.log(response.TableNames.join('\n'));
  })
  .catch((error) => {
    console.error(error);
  });
```

Koordinieren Sie mehrere Versprechen

In einigen Fällen muss Ihr Code mehrere asynchrone Aufrufe ausführen, die nur dann eine Aktion erfordern, wenn alle erfolgreich zurückgegeben wurden. Wenn Sie diese einzelnen asynchronen Methodenaufrufe mit Promises verwalten, können Sie ein zusätzliches Promise erstellen, das die `all`-Methode verwendet.

Diese Methode erfüllt dieses übergeordnete Promise-Objekt, falls und wenn das an die Methode übergebene Array der Promises erfüllt ist. Die Callback-Funktion wird als Array von Werten der Promises übergeben, die an die `all`-Methode übergeben werden.

Im folgenden Beispiel muss eine AWS Lambda Funktion drei asynchrone Aufrufe an Amazon DynamoDB tätigen, kann aber erst abgeschlossen werden, nachdem die Zusagen für jeden Aufruf erfüllt wurden.

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

console.log("Value 0 is " + values[0].toString());
console.log("Value 1 is " + values[1].toString());
console.log("Value 2 is " + values[2].toString());

return values;
```

Browser- und Node.js unterstützen Promises

Die Support für native JavaScript Promises (ECMAScript 2015) hängt von der JavaScript Engine und Version ab, in der Ihr Code ausgeführt wird. Informationen zur Unterstützung von Promises in jeder Umgebung JavaScript, in der Ihr Code ausgeführt werden muss, finden Sie in der [ECMAScript-Kompatibilitätstabelle](#) unter. GitHub

Verwenden Sie eine anonyme Rückruffunktion

Jede Serviceobjektmethode kann eine anonyme Callback-Funktion als letzten Parameter akzeptieren. Die Signatur dieser Callback-Funktion lautet wie folgt.

```
function(error, data) {  
    // callback handling code  
};
```

Diese Callback-Funktion wird ausgeführt, wenn entweder eine erfolgreiche Antwort oder Fehlerdaten zurückgegeben werden. Wenn der Methodenaufruf erfolgreich war, wird der Callback-Funktion der Inhalt der Antwort im `data`-Parameter bereitgestellt. Wenn der Aufruf nicht erfolgreich war, werden die Details über den Fehler im `error`-Parameter angegeben.

In der Regel prüft der Code innerhalb der Callback-Funktion, ob ein Fehler vorliegt, den er verarbeitet, wenn ein Fehler zurückgegeben wird. Wenn kein Fehler zurückgegeben wird, ruft der Code die Antwortdaten vom `data`-Parameter ab. Die grundlegende Form der Callback-Funktion sieht wie folgt aus.

```
function(error, data) {  
    if (error) {  
        // error handling code  
        console.log(error);  
    } else {  
        // data handling code  
        console.log(data);  
    }  
};
```

Im vorherigen Beispiel werden die Details zum Fehler oder die zurückgegebenen Daten in der Konsole protokolliert. Hier finden Sie ein Beispiel für eine Callback-Funktion, die als Teil des Aufrufs einer Methode in einem Service-Objekt zurückgegeben wird.

```
ec2.describeInstances(function(error, data) {  
    if (error) {  
        console.log(error); // an error occurred  
    } else {  
        console.log(data); // request succeeded  
    }  
});
```

Erstellen Sie Service-Client-Anfragen

Das Stellen von Anfragen an AWS Serviceclients ist unkompliziert. Version 3 (V3) des SDK für JavaScript ermöglicht das Senden von Anfragen.

Note

Sie können Operationen auch mit Befehlen der Version 2 (V2) ausführen, wenn Sie die Version V3 des SDK für verwenden JavaScript. Weitere Informationen finden Sie unter [Verwenden von v2-Befehlen](#).

Um eine Anfrage zu senden:

1. Initialisieren Sie ein Client-Objekt mit der gewünschten Konfiguration, z. B. einer bestimmten AWS Region.
2. (Optional) Erstellen Sie ein JSON-Anforderungsobjekt mit den Werten für die Anfrage, z. B. dem Namen eines bestimmten Amazon S3 S3-Buckets. Sie können die Parameter für die Anfrage überprüfen, indem Sie sich das API-Referenzthema für die Schnittstelle ansehen, deren Name der Client-Methode zugeordnet ist. Wenn Sie beispielsweise die *AbcCommandClient*-Methode verwenden, lautet die Anforderungsschnittstelle *AbcInput*.
3. Initialisieren Sie optional einen Dienstbefehl mit dem Anforderungsobjekt als Eingabe.
4. Rufen Sie send den Client mit dem Befehlsobjekt als Eingabe auf.

Um beispielsweise Ihre Amazon DynamoDB-Tabellen aufzulisten `us-west-2`, können Sie dies mit `async/await` tun.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";

(async function () {
  const dbClient = new DynamoDBClient({ region: 'us-west-2' });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
```

```
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err);
  }
}());
```

Bearbeiten Sie die Antworten der Servicekunden

Nachdem eine Dienstclient-Methode aufgerufen wurde, gibt sie eine Antwortobjektinstanz einer Schnittstelle zurück, deren Name der Client-Methode zugeordnet ist. Wenn Sie beispielsweise die `AbcCommandClient`-Methode verwenden, ist das Antwortobjekt vom Typ `AbcResponse` (Schnittstelle).

In der Antwort zurückgegebene Zugriffsdaten

Das Antwortobjekt enthält die Daten als Eigenschaften, die von der Serviceanfrage zurückgegeben wurden.

[Erstellen Sie Service-Client-Anfragen](#) In gab der `ListTablesCommand` Befehl die Tabellennamen in der `TableNames` Eigenschaft der Antwort zurück.

Auf Fehlerinformationen zugreifen

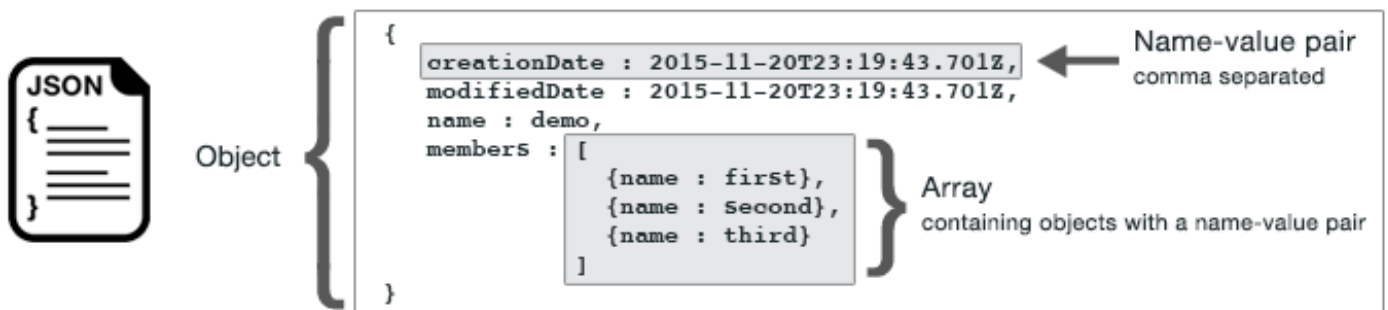
Wenn ein Befehl fehlschlägt, wird eine Ausnahme ausgelöst. Der folgende Codeausschnitt zeigt, wie eine Serviceausnahme behandelt werden kann.

```
try {
  await client.send(someCommand);
} catch (e) {
  if (e.name === "InvalidSignatureException") {
    // Handle InvalidSignatureException
  } else if (e.name === "ResourceNotFoundException") {
    // Handle ResourceNotFoundException
  } else if (e.name === "FooServiceException") {
    // Handle all other server-side exceptions from Foo service
  } else {
    // Handle errors from SDK
  }
}
```

Arbeiten Sie mit JSON

JSON ist ein Format für den Datenaustausch, das sowohl für Menschen als auch für Maschinen lesbar ist. Obwohl der Name JSON eine Abkürzung für JavaScript Object Notation ist, ist das Format von JSON unabhängig von jeder Programmiersprache.

The AWS SDK for JavaScript verwendet JSON, um bei Anfragen Daten an Serviceobjekte zu senden, und empfängt Daten von Serviceobjekten als JSON. Weitere Informationen über JSON finden Sie auf der Website json.org.



JSON stellt Daten auf zwei Arten dar:

- Als Objekt, das eine ungeordnete Sammlung von Name-Wert-Paaren ist. Ein Objekt wird innerhalb von zwei Klammern ({ und }) definiert. Jedes Name-Wert-Paar beginnt mit dem Namen, gefolgt von einem Doppelpunkt und dem Wert. Name-Wert-Paare werden durch Kommas voneinander getrennt.
- Als Array, das eine geordnete Sammlung von Werten ist. Ein Array wird innerhalb von zwei Klammern ([und]) definiert. Elemente im Array werden durch Kommas voneinander getrennt.

Hier sehen Sie ein Beispiel für ein JSON-Objekt mit einem Array von Objekten, in dem die Objekte Karten in einem Kartenspiel darstellen. Jede Karte wird durch zwei Namen-Wert-Paare definiert. Eines davon gibt einen eindeutigen Wert an, um die Karte zu identifizieren, und das andere gibt die URL an, die auf das Kartenbild verweist.

```
var cards = [
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}
]
```

```
];
```

JSON als Serviceobjektparameter

Hier ist ein Beispiel für einfaches JSON, das verwendet wird, um die Parameter eines Aufrufs eines AWS Lambda Serviceobjekts zu definieren.

```
const params = {
  FunctionName : funcName,
  Payload : JSON.stringify(payload),
  LogType : LogType.Tail,
};
```

Das `params`-Objekt wird durch drei Name-Wert-Paare definiert, die durch Kommas innerhalb einer linken und einer rechten Klammer getrennt sind. Beim Bereitstellen von Parametern für einen Service-Objektmethodenaufruf werden die Namen durch die Parameternamen für die Service-Objektmethode bestimmt, die Sie aufrufen möchten. Beim Aufrufen einer Lambda-Funktion sind, und die Parameter `FunctionName`, `Payload`, die verwendet `LogType` werden, um die `invoke` Methode für ein Lambda-Serviceobjekt aufzurufen.

Wenn Sie Parameter an einen Methodenaufruf eines Serviceobjekts übergeben, stellen Sie das JSON-Objekt für den Methodenaufruf bereit, wie im folgenden Beispiel für den Aufruf einer Lambda-Funktion gezeigt.

```
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 };
};
```

SDK für JavaScript Codebeispiele

Die Themen in diesem Abschnitt enthalten Beispiele für die AWS SDK for JavaScript Verwendung der APIs verschiedener Dienste zur Ausführung allgemeiner Aufgaben.

Den Quellcode für diese und andere Beispiele finden Sie im [AWS Codebeispiel-Repository unter GitHub](#). Um dem AWS Dokumentationsteam ein neues Codebeispiel vorzuschlagen, das es erstellen könnte, erstellen Sie eine Anfrage. Das Team möchte Codebeispiele erstellen, die breitere Szenarien und Anwendungsfälle abdecken, im Vergleich zu einfachen Codeausschnitten, die nur einzelne API-Aufrufe abdecken. Eine Anleitung dazu findest du im Abschnitt „Code erstellen“ in den [Richtlinien für Beiträge](#) von GitHub.

Wichtig

In diesen Beispielen wird die ECMAScript6-Import-/Export-Syntax verwendet.

- Dies erfordert Node.js Version 14.17 oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie [JavaScript ES6/CommonJS-Syntax](#) die Konvertierungsrichtlinien.

Themen

- [JavaScript ES6/CommonJS-Syntax](#)
- [Beispiele für Amazon DynamoDB](#)
- [Beispiele für AWS Elemental MediaConvert](#)
- [Beispiele für AWS Lambda](#)
- [Amazon Lex Beispiele](#)
- [Beispiele für Amazon Polly](#)
- [Beispiele für Amazon Redshift](#)
- [Beispiele für Amazon Simple Email Service](#)
- [Beispiele für Amazon Simple Notification Service](#)
- [Beispiele für Amazon Transcribe](#)
- [Node.js auf einer Amazon EC2 EC2-Instance einrichten](#)
- [Erstellen Sie eine App zum Senden von Daten an DynamoDB](#)

- [Erstellen Sie eine Transkriptions-App mit authentifizierten Benutzern](#)
- [Lambda mit API Gateway aufrufen](#)
- [AWSServerlose Workflows erstellen mit AWS SDK for JavaScript](#)
- [Erstellen geplanter Ereignisse zum Ausführen von -AWS LambdaFunktionen](#)
- [Erstellen eines Amazon Lex-Chatbots](#)
- [Erstellen einer Beispiel-Messaging-Anwendung](#)

JavaScript ES6/CommonJS-Syntax

DieAWS SDK for JavaScript Codebeispiele sind in ECMAScript 6 (ES6) geschrieben. ES6 bietet neue Syntax und neue Funktionen, um Ihren Code moderner und lesbarer zu machen und mehr zu tun.

ES6 erfordert, dass Sie Node.js Version 13.x oder höher verwenden. Informationen zum Herunterladen der neuesten Version von Node.js finden Sie unter [Node.js Downloads](#). Wenn Sie möchten, können Sie jedes unserer Beispiele mit einer CommonJS-Syntax konvertieren. Gehen Sie dazu wie nachfolgend beschrieben vor.

- Entfernen Sie "type" : "module" aus derpackage.json in Ihrer Projektumgebung.
- Konvertiert alleimport ES6-Anweisungen inrequire CommonJS-Anweisungen. Konvertiere zum Beispiel:

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";
```

Zu seinem CommonJS-Äquivalent:

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");
```

- Konvertiert alleexport ES6-Anweisungen inmodule.exports CommonJS-Anweisungen. Konvertiere zum Beispiel:

```
export {s3}
```

Zu seinem CommonJS-Äquivalent:

```
module.exports = {s3}
```

Das folgende Beispiel zeigt das Codebeispiel für die Erstellung eines Amazon S3 S3-Buckets sowohl in ES6 als auch in CommonJS.

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("../libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
```



```
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
module.exports = {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");

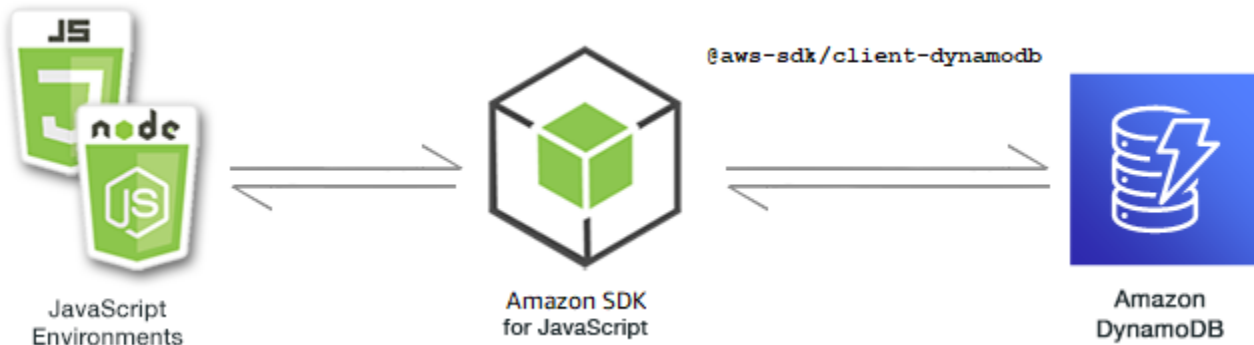
// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

Beispiele für Amazon DynamoDB

Amazon DynamoDB ist eine vollständig verwaltete NoSQL-Cloud-Datenbank, die sowohl Dokument- als auch Key-Value-Speichermodelle unterstützt. Sie erstellen schemalose Tabellen für Daten – ohne Bereitstellung oder Wartung dedizierter Datenbankserver.



Die JavaScript API für DynamoDB wird über die `DynamoDB.DocumentClient` Clientklassen `DynamoDB.DynamoDBStreams`, und verfügbar gemacht. Weitere Informationen zur Verwendung der DynamoDB-Clientklassen finden Sie unter [Class: DynamoDB](#), [Class: DynamoDBStreams](#) und [Class: DynamoDB-Utility](#) in der API-Referenz.

Themen

- [Tabellen in DynamoDB erstellen und verwenden](#)
- [Lesen und Schreiben eines einzelnen Elements in DynamoDB](#)
- [Lesen und Schreiben von Elementen im Batch in DynamoDB](#)
- [Abfragen und Scannen einer DynamoDB-Tabelle](#)
- [Verwenden des DynamoDB-Dokumentenclients](#)

Tabellen in DynamoDB erstellen und verwenden



Dieses Node.js-Codebeispiel zeigt:

- So erstellen und verwalten Sie Tabellen, die zum Speichern und Abrufen von Daten aus DynamoDB verwendet werden.

Das Szenario

Ebenso wie andere Datenbanksysteme speichert auch DynamoDB Daten in Tabellen. Eine DynamoDB-Tabelle ist eine Sammlung von Daten, die in Elementen organisiert sind, die Zeilen entsprechen. Um Daten in DynamoDB zu speichern oder darauf zuzugreifen, erstellen Sie Tabellen und arbeiten mit ihnen.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um grundlegende Operationen mit einer DynamoDB-Tabelle durchzuführen. Der Code verwendet das SDK JavaScript , um Tabellen mithilfe der folgenden Methoden der DynamoDB Client-Klasse zu erstellen und mit ihnen zu arbeiten:

- [CreateTableCommand](#)
- [ListTablesCommand](#)
- [DescribeTableCommand](#)
- [DeleteTableCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese Beispiele für Node.js auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Installieren Sie das SDK für den JavaScript DynamoDB-Client. Weitere Informationen finden Sie unter [Was ist neu in Version 3](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen im Referenzhandbuch](#) für AWS SDKs und Tools.

⚠ Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

ℹ Note

Informationen zu den in diesen Beispielen verwendeten Datentypen finden Sie unter [Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB](#).

Erstellen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `create-table.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Erstellen einer Tabelle, das in diesem Beispiel den Namen und Datentyp für jedes Attribut, das Schlüsselschema, den Namen der Tabelle und die Durchsatzeinheiten beinhaltet, die bereitgestellt werden sollen. Rufen Sie die `CreateTableCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
```

```
        AttributeType: "S",
    },
],
KeySchema: [
    {
        AttributeName: "DrinkName",
        KeyType: "HASH",
    },
],
ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
},
});

const response = await client.send(command);
console.log(response);
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node create-table.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Listet Ihre Tabellen auf

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-tables.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Auflisten Ihrer Tabellen, das in diesem Beispiel die Anzahl der aufgeführten Tabellen auf 10 begrenzt. Rufen Sie die `ListTablesCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
    const command = new ListTablesCommand({});
```

```
const response = await client.send(command);
console.log(response);
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-tables.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Beschreiben einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `describe-table.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt, das die Parameter enthält, die zur Beschreibung einer `DescribeTableCommand` Methode des DynamoDB-Dienstobjekts erforderlich sind.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node describe-table.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Löschen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `delete-table.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Löschen einer Tabelle, das in diesem Beispiel den Namen der Tabelle beinhaltet, der als Befehlszeilenparameter bereitgestellt wird. Rufen Sie die `DeleteTableCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node delete-table.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Lesen und Schreiben eines einzelnen Elements in DynamoDB



Dieses Node.js-Codebeispiel zeigt:

- So fügen Sie ein Element zu einer DynamoDB-Tabelle hinzu.
- So rufen Sie ein Element in einer DynamoDB-Tabelle ab.

- So löschen Sie ein Element in einer DynamoDB-Tabelle.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um ein Element in einer DynamoDB-Tabelle zu lesen und zu schreiben, indem Sie die folgenden Methoden der DynamoDB Client-Klasse verwenden:

- [PutItemCommand](#)
- [UpdateItemCommand](#)
- [GetItemCommand](#)
- [DeleteItemCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese Beispiele für Node.js auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter [Tabellen in DynamoDB erstellen und verwenden](#)

Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Note

Informationen zu den in diesen Beispielen verwendeten Datentypen finden Sie unter [Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB](#).

Schreiben eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `put-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Hinzufügen eines Elements, das in diesem Beispiel den Namen der Tabelle und eine Map beinhaltet, die die festzulegenden Attribute und Werte für jedes Attribut definiert. Rufen Sie die `PutItemCommand` Methode des DynamoDB-Client-Dienstobjekts auf.

```
import { PutItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new PutItemCommand({
    TableName: "Cookies",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Item: {
      Flavor: { S: "Chocolate Chip" },
      Variants: { SS: ["White Chocolate Chip", "Chocolate Chunk"] },
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node put-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Aktualisieren eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `update-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt.

Erstellen Sie ein JSON-Objekt mit den Parametern, die zum Hinzufügen eines Elements erforderlich sind. In diesem Beispiel umfasst es den Namen der Tabelle, den zu aktualisierenden Schlüssel und den Datumsausdruck, der die neuen Attributnamen zuordnet, sowie Werte für jedes neue Attribut.

Rufen Sie die `UpdateItemCommand` Methode des DynamoDB-Client-Dienstobjekts auf.

```
import { UpdateItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new UpdateItemCommand({
    TableName: "IceCreams",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Flavor: { S: "Vanilla" },
    },
    UpdateExpression: "set HasChunks = :chunks",
    ExpressionAttributeValues: {
      ":chunks": { BOOL: "false" },
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node update-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abrufen eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Zum Identifizieren des abzurufenden Elements müssen Sie den Wert des Primärschlüssels für dieses Element in der Tabelle angeben. Die `GetItemCommand`-Methode gibt standardmäßig alle Attributwerte zurück, die für das Element definiert sind. Um nur eine Teilmenge aller möglichen Attributwerte abzurufen, geben Sie einen Projektionsausdruck an.

Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Elements, das in diesem Beispiel den Namen der Tabelle, den Namen und Wert des Schlüssels für das abzurufende Element und einen Projektionsausdruck beinhaltet, der das abzurufende Elementattribut identifiziert. Rufen Sie die `GetItemCommand` Methode des DynamoDB-Client-Dienstobjekts auf.

Das folgende Codebeispiel ruft ein Element aus einer Tabelle mit einem Primärschlüssel ab, der nur aus einem Partitionsschlüssel und nicht aus einem Partitions- und einem Sortierschlüssel gleichzeitig besteht. Wenn die Tabelle einen Primärschlüssel hat, der aus einem Partitionsschlüssel und einem Sortierschlüssel besteht, müssen Sie auch den Namen und das Attribut des Sortierschlüssels angeben.

```
import { GetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new GetItemCommand({
    TableName: "CafeTreats",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      TreatId: { N: "101" },
    },
  });
```

```
});

const response = await client.send(command);
console.log(response);
return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node get-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Löschen eines Elements

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `delete-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern zum Löschen eines Elements, das in diesem Beispiel den Namen der Tabelle und den Schlüsselnamen sowie den Wert für das zu löschende Element beinhaltet. Rufen Sie die `DeleteItemCommand` Methode des DynamoDB-Client-Dienstobjekts auf.

```
import { DeleteItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteItemCommand({
    TableName: "Drinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    Key: {
      Name: { S: "Pumpkin Spice Latte" },
    },
  });

  const response = await client.send(command);
  console.log(response);
};
```

```
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node delete-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Lesen und Schreiben von Elementen im Batch in DynamoDB



Dieses Node.js-Codebeispiel zeigt:

- So lesen und schreiben Sie Stapel von Elementen in einer DynamoDB-Tabelle.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um einen Stapel von Elementen in eine DynamoDB-Tabelle einzufügen und einen Stapel von Elementen zu lesen. Der Code verwendet das SDK für JavaScript die Ausführung von Batch-Lese- und Schreiboperationen mit den folgenden Methoden der DynamoDB-Clientklasse:

- [BatchGetItemCommand](#)
- [BatchWriteItemCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen

finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter [Tabellen in DynamoDB erstellen und verwenden](#)

Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Note

Informationen zu den in diesen Beispielen verwendeten Datentypen finden Sie unter [Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB](#).

Elemente in einem Stapel lesen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `batch-get-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abrufen eines Stapels von Elementen, das in diesem Beispiel den Namen einer oder mehrerer zu lesender Tabellen, die in jeder Tabelle zu lesenden Werte von Schlüsseln und den Projektionsausdruck beinhaltet, der die zurückzugebenden Attribute angibt. Rufen Sie die `BatchGetItemCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { BatchGetItemCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchGetItemCommand({
```

```
RequestItems: {
  // Each key in this object is the name of a table. This example refers
  // to a PageAnalytics table.
  PageAnalytics: {
    // Each entry in Keys is an object that specifies a primary key.
    Keys: [
      {
        // "PageName" is the partition key (simple primary key).
        // "S" specifies a string as the data type for the value "Home".
        // For more information about data types,
        // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
        // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
        // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        PageName: { S: "Home" },
      },
      {
        PageName: { S: "About" },
      },
    ],
    // Only return the "PageName" and "PageViews" attributes.
    ProjectionExpression: "PageName, PageViews",
  },
},
});

const response = await client.send(command);
console.log(response.Responses["PageAnalytics"]);
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node batch-get-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Artikel in einem Stapel schreiben

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `batch-write-item.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB

Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den Parametern, die zum Abrufen eines Batches von Elementen erforderlich sind. In diesem Beispiel umfasst dies die Tabelle, in die Sie Elemente schreiben möchten, die Schlüssel, die Sie für jedes Element schreiben möchten, und die Attribute zusammen mit ihren Werten. Rufen Sie die `BatchWriteItemCommand` Methode des `DynamoDB`-Dienstobjekts auf.

```
import {
  BatchWriteItemCommand,
  DynamoDBClient,
} from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new BatchWriteItemCommand({
    RequestItems: {
      // Each key in this object is the name of a table. This example refers
      // to a Coffees table.
      Coffees: [
        // Each entry in Coffees is an object that defines either a PutRequest or
        // DeleteRequest.
        {
          // Each PutRequest object defines one item to be inserted into the table.
          PutRequest: {
            // The keys of Item are attribute names. Each attribute value is an object
            // with a data type and value.
            // For more information about data types,
            // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes
            Item: {
              Name: { S: "Donkey Kick" },
              Process: { S: "Wet-Hulled" },
              Flavors: { SS: ["Earth", "Syrup", "Spice"] },
            },
          },
        },
        {
          PutRequest: {
            Item: {
              Name: { S: "Flora Ethiopia" },
              Process: { S: "Washed" },
              Flavors: { SS: ["Stone Fruit", "Toasted Almond", "Delicate"] },
            },
          },
        },
      ],
    },
  });
};
```



```
    },  
    },  
  ],  
},  
});  
  
const response = await client.send(command);  
console.log(response);  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node batch-write-item.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abfragen und Scannen einer DynamoDB-Tabelle



Dieses Node.js-Codebeispiel zeigt:

- So fragen Sie eine DynamoDB-Tabelle ab und scannen sie nach Elementen.

Das Szenario

Die Abfrage findet Elemente in einer Tabelle oder einem Sekundärindex nur über Primärschlüsselattributwerte. Sie müssen einen Partitionsschlüsselnamen und einen Wert angeben, nach dem gesucht werden soll. Sie können auch einen Sortierschlüsselnamen und Wert angeben, und einen Vergleichsoperator verwenden, um die Suchergebnisse zu verfeinern. Das Scannen findet Elemente, indem jedes Element in der angegebenen Tabelle überprüft wird.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um ein oder mehrere Elemente zu identifizieren, die Sie aus einer DynamoDB-Tabelle abrufen möchten. Der Code verwendet das SDK JavaScript zum Abfragen und Scannen von Tabellen mithilfe der folgenden Methoden der DynamoDB-Clientklasse:

- [QueryCommand](#)
- [ScanCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese Beispiele für Node.js auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle finden Sie unter [Tabellen in DynamoDB erstellen und verwenden](#)

Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Note

Informationen zu den in diesen Beispielen verwendeten Datentypen finden Sie unter [Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB](#).

Abfragen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `query.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen

Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abfragen der Tabelle, das in diesem Beispiel den Tabellennamen, die von der Abfrage benötigten `ExpressionAttributeValues`, einen `KeyConditionExpression`, der diese Werte verwendet, um zu definieren, welche Elemente die Abfrage zurück gibt, und die Namen der für jedes Element zurückzugebenden Attributwerte beinhaltet. Rufen Sie die `QueryCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new QueryCommand({
    KeyConditionExpression: "Flavor = :flavor",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":flavor": { S: "Key Lime" },
      ":searchKey": { S: "no coloring" },
    },
    FilterExpression: "contains (Description, :searchKey)",
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node query.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Scannen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `scan.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Um auf DynamoDB zuzugreifen, erstellen Sie ein DynamoDB Client-Serviceobjekt. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Scannen der Tabelle nach Elementen, das in diesem Beispiel den Namen der Tabelle, die Liste der Attributwerte, die für jedes übereinstimmende Element zurückgegeben werden sollen, und einen Ausdruck zum Filtern des Ergebnissatzes beinhaltet, um nach Elementen mit einem bestimmten Begriff zu suchen. Rufen Sie die `ScanCommand` Methode des DynamoDB-Dienstobjekts auf.

```
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ScanCommand({
    FilterExpression: "CrustType = :crustType",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    ExpressionAttributeValues: {
      ":crustType": { S: "Graham Cracker" },
    },
    ProjectionExpression: "Flavor, CrustType, Description",
    TableName: "Pies",
  });

  const response = await client.send(command);
  response.Items.forEach(function (pie) {
    console.log(`${pie.Flavor.S} - ${pie.Description.S}\n`);
  });
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node scan.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verwenden des DynamoDB-Dokumentenclients



Dieses Node.js-Codebeispiel zeigt:

- So greifen Sie mit den DynamoDB-Dienstprogrammen auf eine DynamoDB-Tabelle zu.

Das Szenario

Der DynamoDB Document Client vereinfacht die Arbeit mit Elementen, indem er den Begriff der Attributwerte abstrahiert. Diese Abstraktion annotiert native JavaScript Typen, die als Eingabeparameter bereitgestellt werden, und konvertiert kommentierte Antwortdaten in native Typen. JavaScript

Weitere Informationen zum DynamoDB Document Client finden Sie unter [@aws -sdk/lib-dynamodb](#) README auf. GitHub Weitere Informationen zur Programmierung mit Amazon DynamoDB finden Sie unter [Programmieren von Amazon DynamoDB mit JavaScript im Amazon DynamoDB Developer Guide](#).

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um mithilfe von DynamoDB-Dienstprogrammen grundlegende Operationen an einer DynamoDB-Tabelle durchzuführen. Der Code verwendet das SDK JavaScript zum Abfragen und Scannen von Tabellen mithilfe der folgenden Methoden der DynamoDB Document Client-Klasse:

- [GetCommand](#)
- [PutCommand](#)
- [UpdateCommand](#)
- [QueryCommand](#)
- [DeleteCommand](#)

Weitere Informationen zur Konfiguration des DynamoDB Document Client finden Sie unter [@aws -sdk/lib-dynamodb](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese Beispiele für Node.js auszuführen, und installieren Sie die erforderlichen Module und Module von Drittanbietern. AWS SDK for JavaScript Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine DynamoDB-Tabelle, auf deren Elemente Sie zugreifen können. Weitere Hinweise zum Erstellen einer DynamoDB-Tabelle mit dem SDK für JavaScript finden Sie unter [Tabellen in DynamoDB erstellen und verwenden](#) Sie können auch die [DynamoDB-Konsole](#) verwenden, um eine Tabelle zu erstellen.

Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Note

Informationen zu den in diesen Beispielen verwendeten Datentypen finden Sie unter [Unterstützte Datentypen und Benennungsregeln in Amazon DynamoDB](#).

Abrufen eines Elements aus einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Dazu gehört `@aws-sdk/lib-dynamodb` ein Bibliothekspaket, das Dokumenten-Client-Funktionen für `bereitstellt@aws-sdk/client-dynamodb`. Als Nächstes legen Sie die

Konfiguration für Marshalling und Unmarshalling als optionalen zweiten Parameter bei der Erstellung des Dokumentenclients wie unten dargestellt fest. Als Nächstes erstellen Sie die Clients. Erstellen Sie nun ein JSON-Objekt, das die erforderlichen Parameter enthält. Rufen Sie ein Element aus der Tabelle ab. In diesem Beispiel enthält es den Namen der Tabelle, den Namen des Hash-Schlüssels in dieser Tabelle und den Wert des Hash-Schlüssels für das Element, das Sie abrufen möchten. Rufen Sie die `GetCommand` Methode des `DynamoDB Document Client` auf.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node get.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Einfügen eines Elements in eine Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `put.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Dazu gehört `@aws-sdk/lib-dynamodb` ein Bibliothekspaket, das Dokumenten-Client-Funktionen für bereitstellt `@aws-sdk/client-dynamodb`. Als Nächstes legen Sie die Konfiguration für Marshalling und Unmarshalling als optionalen zweiten Parameter bei der Erstellung des Dokumentenclients wie unten dargestellt fest. Als Nächstes erstellen Sie die Clients. Erstellen Sie ein JSON-Objekt mit den Parametern, die zum Schreiben eines Elements in die Tabelle erforderlich sind.

In diesem Beispiel enthält es den Namen der Tabelle und eine Beschreibung des hinzuzufügenden oder zu aktualisierenden Elements, die den Hashkey und den Wert sowie Namen und Werte für Attribute enthält, die für das Element festgelegt werden sollen. Rufen Sie die PutCommand Methode des DynamoDB Document Client auf.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node put.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Aktualisieren eines Elements in einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `update.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Dazu gehört `@aws-sdk/lib-dynamodb` ein Bibliothekspaket, das Dokumenten-Client-Funktionen für bereitstellt `@aws-sdk/client-dynamodb`. Als Nächstes legen Sie die Konfiguration für Marshalling und Unmarshalling als optionalen zweiten Parameter bei der Erstellung des Dokumentenclients wie unten dargestellt fest. Als Nächstes erstellen Sie die Clients. Erstellen Sie ein JSON-Objekt, das die Parameter enthält, die zum Schreiben eines Elements in die Tabelle erforderlich sind. In diesem Beispiel gehören der Name der Tabelle, der Schlüssel des zu aktualisierenden Elements, ein Satz, der die Attribute des UpdateExpressions zu aktualisierenden

Elements mit Tokens definiert, denen Sie in den `ExpressionAttributeValue` Parametern Werte zuweisen. Rufen Sie die `updateCommand` Methode des DynamoDB Document Client auf.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node update.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abfragen einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `query.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Dazu gehört `@aws-sdk/lib-dynamodb` ein Bibliothekspaket, das Dokumenten-Client-Funktionen für bereitstellt `@aws-sdk/client-dynamodb`. Erstellen Sie ein JSON-Objekt mit den erforderlichen Parametern für das Abfragen der Tabelle, das in diesem Beispiel den Tabellennamen, die von der Abfrage benötigten `ExpressionAttributeValues` und einen `KeyConditionExpression` beinhaltet, der diese Werte verwendet, um zu definieren, welche

Elemente die Abfrage zurück gibt. Rufen Sie die `QueryCommand` Methode des `DynamoDB Document Client` auf.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node query.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Löschen eines Elements aus einer Tabelle

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `delete.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Dazu gehört `@aws-sdk/lib-dynamodb` ein Bibliothekspaket, das Dokumenten-Client-Funktionen für bereitstellt `@aws-sdk/client-dynamodb`. Als Nächstes legen Sie die Konfiguration für Marshalling und Unmarshalling als optionalen zweiten Parameter bei der Erstellung des Dokumentenclients wie unten dargestellt fest. Als Nächstes erstellen Sie die Clients. Um auf DynamoDB zuzugreifen, erstellen Sie ein `DynamoDB` Objekt. Erstellen Sie ein JSON-Objekt mit den

Parametern, die zum Löschen eines Elements in der Tabelle erforderlich sind. In diesem Beispiel gehören der Name der Tabelle sowie der Name und der Wert des Hashkeys des Elements, das Sie löschen möchten. Rufen Sie die `DeleteCommand` Methode des DynamoDB Document Client auf.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node delete.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Beispiele für AWS Elemental MediaConvert

AWS Elemental MediaConvert ist ein dateibasierter Transcodierungsservice für Videos, der über Ausstrahlungsfunktionen verfügt. Sie können damit Inhalte für die Übertragung und für die video-on-demand (VOD-) Übertragung über das Internet erstellen. Weitere Informationen finden Sie im [AWS Elemental MediaConvert-Benutzerhandbuch](#).

Die JavaScript API für MediaConvert wird über die MediaConvert Client-Klasse verfügbar gemacht. Weitere Informationen finden Sie unter [Class: MediaConvert](#) in der API-Referenz.

Themen

- [Sie erhalten Ihren regionsspezifischen Endpunkt für MediaConvert](#)

- [Transcodierungsaufträge erstellen und verwalten in MediaConvert](#)
- [Verwenden von Jobvorlagen in MediaConvert](#)

Sie erhalten Ihren regionsspezifischen Endpunkt für MediaConvert



Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie Ihren regionsspezifischen Endpunkt von ab. MediaConvert

Das Szenario

In diesem Beispiel verwenden Sie ein Modul Node.js, um Ihren regionsspezifischen MediaConvert Endpunkt aufzurufen und abzurufen. Sie können Ihre Endpunkt-URL vom Standardendpunkt des Dienstes abrufen und benötigen Ihren regionsspezifischen Endpunkt daher noch nicht. Der Code verwendet das SDK JavaScript , um diesen Endpunkt mithilfe dieser Methode der MediaConvert Client-Klasse abzurufen:

- [DescribeEndpointsCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter [Einrichten von IAM-Berechtigungen](#) im AWS Elemental MediaConvert Benutzerhandbuch.

⚠ Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Ermitteln Sie Ihre Endpunkt-URL

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `emcClientGet.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das MediaConvert Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the AWS Region.
const REGION = "REGION";
//Set the MediaConvert Service Object
const emcClientGet = new MediaConvertClient({ region: REGION });
export { emcClientGet };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_getendpoint.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die leeren Anforderungsparameter für die `DescribeEndpointsCommand` Methode der MediaConvert Clientklasse zu übergeben. Rufen Sie anschließend die Methode `DescribeEndpointsCommand` auf.

```
// Import required AWS-SDK clients and commands for Node.js
import { DescribeEndpointsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClientGet } from "../libs/emcClientGet.js";

//set the parameters.
const params = { MaxResults: 0 };

const run = async () => {
```

```
try {
  // Create a new service object and set MediaConvert to customer endpoint
  const data = await emcClientGet.send(new DescribeEndpointsCommand(params));
  console.log("Your MediaConvert endpoint is ", data.Endpoints);
  return data;
} catch (err) {
  console.log("Error", err);
}
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_getendpoint.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Transcodierungsaufträge erstellen und verwalten in MediaConvert



Dieses Node.js-Codebeispiel zeigt:

- So geben Sie den regionsspezifischen Endpunkt an, mit dem verwendet werden soll. MediaConvert
- So erstellen Sie Transcodierungsaufträge in. MediaConvert
- So stornieren Sie einen Transcodierungsauftrag.
- So rufen Sie die JSON für einen abgeschlossenen Transcodierungsauftrag ab.
- So rufen Sie ein JSON-Array für bis zu 20 der zuletzt erstellten Aufträge ab.

Das Szenario

In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Transcodierungsaufträge MediaConvert zu erstellen und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- [CreateJobCommand](#)

- [CancelJobCommand](#)
- [GetJobCommand](#)
- [ListJobsCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen und konfigurieren Sie Amazon S3 S3-Buckets, die Speicherplatz für Auftragseingabedateien und Ausgabedateien bereitstellen. Einzelheiten finden Sie unter [Speicher für Dateien erstellen](#) im AWS Elemental MediaConvertBenutzerhandbuch.
- Laden Sie das Eingabevideo in den Amazon S3 S3-Bucket hoch, den Sie für den Eingabespeicher bereitgestellt haben. Eine Liste der unterstützten Eingabe-Videocodecs und Container finden Sie im Benutzerhandbuch unter [Unterstützte Eingabecodecs und Container](#). AWS Elemental MediaConvert
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter [Einrichten von IAM-Berechtigungen](#) im AWS Elemental MediaConvert Benutzerhandbuch.

Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Konfigurieren des SDKs

Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Da für jedes Konto benutzerdefinierte Endpunkte MediaConvert verwendet werden, müssen Sie auch die MediaConvert Client-Klasse so konfigurieren, dass sie Ihren regionspezifischen Endpunkt verwendet. Setzen Sie zu diesem Zweck den `endpoint`-Parameter auf `mediaconvert(endpoint)`.

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";
```

Definition eines einfachen Transcodierungsauftrags

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das MediaConvert Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region. Ersetzen Sie **ENDPOINT** durch Ihren MediaConvert Kontoendpunkt. Dies können Sie auf der Kontoseite in der MediaConvert Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `emc_createjob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie die JSON für die Definition der Transcodierungsauftragsparameter.

Diese Parameter sind sehr detailliert. Sie können die [AWS Elemental MediaConvertKonsole](#) verwenden, um die JSON-Jobparameter zu generieren, indem Sie Ihre Job-Einstellungen in der Konsole auswählen und dann am Ende des Job-Abschnitts die Option `Job-JSON anzeigen wählen`. Dieses Beispiel enthält die JSON für einen einfachen Auftrag.

Note

Ersetzen Sie `JOB_QUEUE_ARN` durch die MediaConvert Job-Warteschlange, `IAM_ROLE_ARN` durch den Amazon-Ressourcennamen (ARN) der IAM-Rolle, `OUTPUT_BUCKET_NAME` durch den Ziel-Bucket-Namen, zum Beispiel `"s3://OUTPUT_BUCKET_NAME/"`, und `INPUT_BUCKET_AND_FILENAME` durch den Eingabe-Bucket und den Dateinamen, zum Beispiel `"s3://INPUT_BUCKET/FILE_NAME"`.

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
BUCKET_NAME/"
          },
        },
      },
    ],
    Outputs: [
      {
        VideoDescription: {
          ScalingBehavior: "DEFAULT",
          TimecodeInsertion: "DISABLED",
          AntiAlias: "ENABLED",
          Sharpness: 50,
          CodecSettings: {
            Codec: "H_264",
            H264Settings: {
              InterlaceMode: "PROGRESSIVE",
              NumberReferenceFrames: 3,
              Syntax: "DEFAULT",
              Softness: 0,
            },
          },
        },
      },
    ],
  },
}
```

```
GopClosedCadence: 1,
GopSize: 90,
Slices: 1,
GopBReference: "DISABLED",
SlowPal: "DISABLED",
SpatialAdaptiveQuantization: "ENABLED",
TemporalAdaptiveQuantization: "ENABLED",
FlickerAdaptiveQuantization: "DISABLED",
EntropyEncoding: "CABAC",
Bitrate: 5000000,
FramerateControl: "SPECIFIED",
RateControlMode: "CBR",
CodecProfile: "MAIN",
Telecine: "NONE",
MinIInterval: 0,
AdaptiveQuantization: "HIGH",
CodecLevel: "AUTO",
FieldEncoding: "PAFF",
SceneChangeDetect: "ENABLED",
QualityTuningLevel: "SINGLE_PASS",
FramerateConversionAlgorithm: "DUPLICATE_DROP",
UnregisteredSeiTimecode: "DISABLED",
GopSizeUnits: "FRAMES",
ParControl: "SPECIFIED",
NumberBFramesBetweenReferenceFrames: 2,
RepeatPps: "DISABLED",
FramerateNumerator: 30,
FramerateDenominator: 1,
ParNumerator: 1,
ParDenominator: 1,
},
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
  {
    AudioTypeControl: "FOLLOW_INPUT",
    CodecSettings: {
      Codec: "AAC",
      AacSettings: {
        AudioDescriptionBroadcasterMix: "NORMAL",
```

```
        RateControlMode: "CBR",
        CodecProfile: "LC",
        CodingMode: "CODING_MODE_2_0",
        RawFormat: "NONE",
        SampleRate: 48000,
        Specification: "MPEG4",
        Bitrate: 64000,
    },
},
LanguageCodeControl: "FOLLOW_INPUT",
AudioSourceName: "Audio Selector 1",
},
],
ContainerSettings: {
    Container: "MP4",
    Mp4Settings: {
        CslgAtom: "INCLUDE",
        FreeSpaceBox: "EXCLUDE",
        MoovPlacement: "PROGRESSIVE_DOWNLOAD",
    },
},
NameModifier: "_1",
},
],
},
],
AdAvailOffset: 0,
Inputs: [
    {
        AudioSelectors: {
            "Audio Selector 1": {
                Offset: 0,
                DefaultSelection: "NOT_DEFAULT",
                ProgramSelection: 1,
                SelectorType: "TRACK",
                Tracks: [1],
            },
},
        VideoSelector: {
            ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
```

```
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
    TimecodeSource: "EMBEDDED",
    FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
"s3://BUCKET_NAME/FILE_NAME"
  },
],
TimecodeConfig: {
  Source: "EMBEDDED",
},
},
};
```

Einen Transcodierungsauftrag erstellen

Rufen Sie nach dem Erstellen der JSON-Jobparameter die asynchrone `run` Methode auf, um ein `MediaConvert Client`-Dienstobjekt aufzurufen, und übergeben Sie dabei die Parameter. Die ID des erstellten Auftrags wird in den `data` der Antwort zurückgegeben.

```
const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_createjob.js
```

Den vollständigen Beispielcode finden Sie [hier unter GitHub](#).

Abbrechen eines Transcodierungsauftrags

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das `MediaConvert Client`-Objekt

erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region. Ersetzen Sie **ENDPOINT** durch Ihren MediaConvert Kontoendpunkt. Dies können Sie auf der Kontoseite in der MediaConvert Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_canceljob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Erstellen Sie die JSON, die die ID des zu stornierenden Auftrags enthält. Rufen Sie dann die `CancelJobCommand` Methode auf, indem Sie ein Versprechen zum Aufrufen eines MediaConvert Client-Dienstobjekts erstellen und die Parameter übergeben. Verarbeiten Sie die Antwort im Promise-Rückruf.

Note

Ersetzen Sie **JOB_ID** durch die ID des Jobs, der abgebrochen werden soll.

```
// Import required AWS-SDK clients and commands for Node.js
import { CancelJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID

const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log("Job " + params.Id + " is canceled");
    return data;
  } catch (err) {
    console.log("Error", err);
  }
}
```

```
};  
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node ec2_canceljob.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Liste der letzten Transcodierungsaufträge

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das `MediaConvert Client`-Objekt erstellt wird. Ersetzen Sie `REGION` durch Ihre AWS Region. Ersetzen Sie `ENDPOINT` durch Ihren `MediaConvert` Kontoendpunkt. Dies können Sie auf der Kontoseite in der `MediaConvert` Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";  
// Set the account end point.  
const ENDPOINT = {  
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",  
};  
// Set the MediaConvert Service Object  
const emcClient = new MediaConvertClient(ENDPOINT);  
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `emc_listjobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie die `JSON`-Parameter, einschließlich Werten, um anzugeben `ASCENDING`, ob `DESCENDING` die Liste nach dem Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange und dem Status der einzubeziehenden Jobs sortiert werden soll. Rufen Sie dann die `ListJobsCommand` Methode auf, indem Sie eine Zusage für den Aufruf eines `MediaConvert Client-Serviceobjekts` erstellen und die Parameter übergeben.

Note

Ersetzen Sie *QUEUE_ARN* durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange und *STATUS* durch den Status der Warteschlange.

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobsCommand(params));
    console.log("Success. Jobs: ", data.Jobs);
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_listjobs.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verwenden von Jobvorlagen in MediaConvert



Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie AWS Elemental MediaConvert-Auftragsvorlagen.
- So verwenden Sie eine Auftragsvorlage zum Erstellen eines Transcodierungsauftrags.
- So listen Sie alle Ihre Auftragsvorlagen auf.
- So löschen Sie Auftragsvorlagen.

Das Szenario

Die JSON-Datei, die für die Erstellung eines Transcodierungsauftrags erforderlich MediaConvert ist, ist detailliert und enthält eine große Anzahl von Einstellungen. Sie können die Auftragserstellung erheblich vereinfachen, indem Sie zweifelsfrei funktionierende Einstellungen in einer Auftragsvorlage speichern, die zum Erstellen nachfolgender Aufträge verwendet werden kann. In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Jobvorlagen MediaConvert zu erstellen, zu verwenden und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- [CreateJobTemplateCommand](#)
- [CreateJobCommand](#)
- [DeleteJobTemplateCommand](#)
- [ListJobTemplatesCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter [Einrichten von IAM-Berechtigungen](#) im AWS Elemental MediaConvert Benutzerhandbuch.

⚠ Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Eine Jobvorlage erstellen

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das MediaConvert Client-Objekt erstellt wird. Ersetzen Sie `REGION` durch Ihre AWS Region. Ersetzen Sie `ENDPOINT` durch Ihren MediaConvert Kontoendpunkt. Dies können Sie auf der Kontoseite in der MediaConvert Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `emc_create_jobtemplate.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Geben Sie die Parameter-JSON für die Vorlagenerstellung an. Sie können die meisten JSON-Parameter aus einem vorherigen erfolgreichen Job verwenden, um die Settings-Werte in der Vorlage anzugeben. In diesem Beispiel werden die Aufgabeneinstellungen aus [Transcodierungsaufträge erstellen und verwalten in MediaConvert](#) verwendet.

Rufen Sie die `CreateJobTemplateCommand` Methode auf, indem Sie ein Versprechen für den Aufruf eines MediaConvert Client-Dienstobjekts erstellen und die Parameter übergeben.

Note

Ersetzen Sie *JOB_QUEUE_ARN* durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange und *BUCKET_NAME* durch den Namen des Amazon S3 S3-Ziel-Buckets — zum Beispiel "s3://BUCKET_NAME/".

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

const params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
          },
        },
      },
    ],
    Outputs: [
      {
        VideoDescription: {
          ScalingBehavior: "DEFAULT",
          TimecodeInsertion: "DISABLED",
          AntiAlias: "ENABLED",
          Sharpness: 50,
          CodecSettings: {
            Codec: "H_264",
            H264Settings: {
              InterlaceMode: "PROGRESSIVE",
              NumberReferenceFrames: 3,
              Syntax: "DEFAULT",
              Softness: 0,
              GopClosedCadence: 1,
              GopSize: 90,
            },
          },
        },
      },
    ],
  },
};
```

```
Slices: 1,
GopBReference: "DISABLED",
SlowPal: "DISABLED",
SpatialAdaptiveQuantization: "ENABLED",
TemporalAdaptiveQuantization: "ENABLED",
FlickerAdaptiveQuantization: "DISABLED",
EntropyEncoding: "CABAC",
Bitrate: 5000000,
FramerateControl: "SPECIFIED",
RateControlMode: "CBR",
CodecProfile: "MAIN",
Telecine: "NONE",
MinIInterval: 0,
AdaptiveQuantization: "HIGH",
CodecLevel: "AUTO",
FieldEncoding: "PAFF",
SceneChangeDetect: "ENABLED",
QualityTuningLevel: "SINGLE_PASS",
FramerateConversionAlgorithm: "DUPLICATE_DROP",
UnregisteredSeiTimecode: "DISABLED",
GopSizeUnits: "FRAMES",
ParControl: "SPECIFIED",
NumberBFramesBetweenReferenceFrames: 2,
RepeatPps: "DISABLED",
FramerateNumerator: 30,
FramerateDenominator: 1,
ParNumerator: 1,
ParDenominator: 1,
},
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
  {
    AudioTypeControl: "FOLLOW_INPUT",
    CodecSettings: {
      Codec: "AAC",
      AacSettings: {
        AudioDescriptionBroadcasterMix: "NORMAL",
        RateControlMode: "CBR",
        CodecProfile: "LC",
```

```
        CodingMode: "CODING_MODE_2_0",
        RawFormat: "NONE",
        SampleRate: 48000,
        Specification: "MPEG4",
        Bitrate: 64000,
      },
    ],
    LanguageCodeControl: "FOLLOW_INPUT",
    AudioSourceName: "Audio Selector 1",
  },
  ContainerSettings: {
    Container: "MP4",
    Mp4Settings: {
      CslgAtom: "INCLUDE",
      FreeSpaceBox: "EXCLUDE",
      MoovPlacement: "PROGRESSIVE_DOWNLOAD",
    },
  },
  NameModifier: "_1",
},
],
},
],
AdAvailOffset: 0,
Inputs: [
  {
    AudioSelectors: {
      "Audio Selector 1": {
        Offset: 0,
        DefaultSelection: "NOT_DEFAULT",
        ProgramSelection: 1,
        SelectorType: "TRACK",
        Tracks: [1],
      },
    },
    VideoSelector: {
      ColorSpace: "FOLLOW",
    },
    FilterEnable: "AUTO",
    PsiControl: "USE_PSI",
    FilterStrength: 0,
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
  }
]
```

```
        TimecodeSource: "EMBEDDED",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};

const run = async () => {
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_create_jobtemplate.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Einen Transcodierungsauftrag aus einer Jobvorlage erstellen

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das MediaConvert Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region. Ersetzen Sie **ENDPOINT** durch Ihren MediaConvert Kontoendpunkt. Dies können Sie auf der Kontoseite in der MediaConvert Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_template_createjob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie die Auftragserstellungsparameter-JSON, einschließlich dem Namen der zu verwendenden Auftragsvorlage und der zu verwendenden Settings, die spezifisch für den Auftrag sind, den Sie erstellen. Rufen Sie dann die `CreateJobsCommand` Methode auf, indem Sie ein Versprechen zum Aufrufen eines `MediaConvert` Client-Dienstobjekts erstellen und die Parameter übergeben.

Note

Ersetzen Sie *`JOB_QUEUE_ARN`* durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange, *`KEY_PAIR_NAME`* durch, *`TEMPLATE_NAME`* durch, *`ROLE_ARN`* durch den Amazon-Ressourcennamen (ARN) der Rolle und *`INPUT_BUCKET_AND_FILENAME`* durch den *Eingabe-Bucket* und den Dateinamen — zum Beispiel `"s3://BUCKET_NAME/FILE_NAME"`.

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
      },
    ],
  },
};
```

```

    VideoSelector: {
      ColorSpace: "FOLLOW",
    },
    FilterEnable: "AUTO",
    PsiControl: "USE_PSI",
    FilterStrength: 0,
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
    TimecodeSource: "EMBEDDED",
    FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
"s3://BUCKET_NAME/FILE_NAME"
  },
],
},
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_template_createjob.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Listet Ihre Jobvorlagen auf

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das `MediaConvert Client`-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region. Ersetzen Sie **ENDPOINT** durch Ihren `MediaConvert` Kontoendpunkt. Dies können Sie auf der Kontoseite in der `MediaConvert` Konsole tun.

```

import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {

```

```
    endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
  };
  // Set the MediaConvert Service Object
  const emcClient = new MediaConvertClient(ENDPOINT);
  export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_listtemplates.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Anfrageparameter für die `listTemplates`-Methode der `MediaConvert`-Client-Klasse zu übergeben. Schließen Sie Werte ein, um zu bestimmen, welche Vorlagen gelistet werden sollen (`NAME`, `CREATION_DATE`, `SYSTEM`), wie viele und deren Sortierreihenfolge. Um die `ListTemplatesCommand` Methode aufzurufen, erstellen Sie ein Versprechen zum Aufrufen eines `MediaConvert` Client-Dienstobjekts und übergeben dabei die Parameter.

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobTemplatesCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

const params = {
  ListBy: "NAME",
  MaxResults: 10,
  Order: "ASCENDING",
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobTemplatesCommand(params));
    console.log("Success ", data.JobTemplates);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.


```
node emc_listtemplates.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Löschen einer Jobvorlage

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `emcClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das `MediaConvert Client`-Objekt erstellt wird. Ersetzen Sie `REGION` durch Ihre AWS Region. Ersetzen Sie `ENDPOINT` durch Ihren `MediaConvert` Kontoendpunkt. Dies können Sie auf der Kontoseite in der `MediaConvert` Konsole tun.

```
import { MediaConvertClient } from "@aws-sdk/client-mediaconvert";
// Set the account end point.
const ENDPOINT = {
  endpoint: "https://ENDPOINT_UNIQUE_STRING.mediaconvert.REGION.amazonaws.com",
};
// Set the MediaConvert Service Object
const emcClient = new MediaConvertClient(ENDPOINT);
export { emcClient };
```

Diesen Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `emc_delete_template.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den Namen der Auftragsvorlage, die Sie löschen möchten, als Parameter für die `DeleteJobTemplateCommand`-Methode der `MediaConvert`-Client-Klasse zu übergeben.

Um die `DeleteJobTemplateCommand` Methode aufzurufen, erstellen Sie ein Versprechen zum Aufrufen eines `MediaConvert Client`-Dienstobjekts und übergeben dabei die Parameter.

```
// Import required AWS-SDK clients and commands for Node.js
import { DeleteJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

// Set the parameters
const params = { Name: "test" }; //TEMPLATE_NAME

const run = async () => {
  try {
```

```
const data = await emcClient.send(new DeleteJobTemplateCommand(params));
console.log(
  "Success, template deleted! Request ID:",
  data.$metadata.requestId,
);
return data;
} catch (err) {
  console.log("Error", err);
}
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_deletetemplate.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Beispiele für AWS Lambda

AWS Lambda ist ein serverloser Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten, eine Workload-sensitive Cluster-Skalierungslogik zu erstellen, Eventintegrationen zu verwalten oder Laufzeiten zu verwalten.

Die JavaScript API für AWS Lambda wird über die [LambdaServiceClient](#)-Klasse bereitgestellt.

Hier ist eine Liste von Beispielen, die zeigen, wie Lambda-Funktionen mit der AWS SDK for JavaScript v3 erstellt und verwendet werden:

- [Lambda mit API Gateway aufrufen](#)
- [Erstellen geplanter Ereignisse zum Ausführen von -AWS LambdaFunktionen](#)

Amazon Lex Beispiele

Amazon Lex ist ein AWS Service zum Integrieren von Konversationsschnittstellen in Anwendungen mithilfe von Sprache und Text.

Die JavaScript API für Amazon Lex wird über die [Lex Runtime Service-Clientklasse](#) bereitgestellt.

- [Erstellen eines Amazon Lex-Chatbots](#)

Beispiele für Amazon Polly



Dieses Node.js-Codebeispiel zeigt:

- Laden Sie mit Amazon Polly aufgenommene Audiodaten auf Amazon S3 hoch

Das Szenario

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um mit Amazon Polly aufgezeichnetes Audio mithilfe der folgenden Methoden der Amazon S3-Clientklasse automatisch auf Amazon S3 hochzuladen:

- [StartSpeechSynthesisTaskCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie eine Projektumgebung ein, um JavaScript Node-Beispiele auszuführen, indem Sie den Anweisungen unter folgen [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.
- Erstellen Sie eine AWS Identity and Access Management (IAM) unauthentifizierte Amazon Cognito Cognito-Benutzerrolle polly: SynthesizeSpeech permissions und einen Amazon Cognito Cognito-Identitätspool mit der zugehörigen IAM-Rolle. Im folgenden [Erstellen Sie die Ressourcen mit dem AWSAWS CloudFormation](#) Abschnitt wird beschrieben, wie Sie diese Ressourcen erstellen.

Note

In diesem Beispiel wird Amazon Cognito verwendet, aber wenn Sie Amazon Cognito nicht verwenden, muss Ihr AWS Benutzer über die folgenden IAM-Berechtigungsrichtlinien verfügen

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "polly:SynthesizeSpeech",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Erstellen Sie die Ressourcen mit dem AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt zu erstellen und bereitzustellen. [Weitere Informationen zu AWS CloudFormation finden Sie im AWS CloudFormation Benutzerhandbuch.](#)

Um den AWS CloudFormation Stapel zu erstellen:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

Note

Die AWS CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zu finden Sie im [AWS Cloud Development Kit \(AWS CDK\) Entwicklerhandbuch](#). AWS CDK

- Führen Sie den folgenden Befehl von der Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

! Important

Der Stack-Name muss innerhalb einer AWS Region und eines Kontos eindeutig sein. AWS Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLI Befehlsreferenz](#) und im [AWS CloudFormation Benutzerhandbuch](#).

- Navigieren Sie zur AWS CloudFormation Managementkonsole, wählen Sie Stacks, wählen Sie den Stack-Namen und klicken Sie auf die Registerkarte Ressourcen, um eine Liste der erstellten Ressourcen anzuzeigen.

The screenshot shows the AWS CloudFormation console interface. The left sidebar contains navigation options like 'Stacks', 'Stack details', 'StackSets', 'Exports', 'Designer', 'Registry', and 'Feedback'. The main content area is titled 'CloudFormation > Stacks > my-polly-test'. It features a 'Stacks (11)' list with a search filter and a 'View nested' toggle. The 'Resources (5)' tab is active, displaying a table of resources. The table has columns for 'Logical ID', 'Physical ID', and 'Type'. The resources listed are CDKMetadata, CognitoDefaultUnauthenticatedRoleABBF7267, CognitoDefaultUnauthenticatedRoleDefaultPolicy2B700C08, DefaultValid, and ExampleIdentityPool.

Logical ID	Physical ID	Type
CDKMetadata	[REDACTED]	AWS::CDK::Metadata
CognitoDefaultUnauthenticatedRoleABBF7267	my-polly-test-[REDACTED]	AWS::IAM::Role
CognitoDefaultUnauthenticatedRoleDefaultPolicy2B700C08	[REDACTED]	AWS::IAM::Policy
DefaultValid	[REDACTED]	AWS::Cognito::IdentityPoolRole
ExampleIdentityPool	[REDACTED]	AWS::Cognito::IdentityPool

Laden Sie mit Amazon Polly aufgenommene Audiodaten auf Amazon S3 hoch

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `polly_synthesize_to_s3.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie im Code die `REGION` und den `BUCKET_NAME` ein. Um auf Amazon Polly zuzugreifen, erstellen Sie ein Polly Kundenservice-Objekt. Ersetzen Sie „`IDENTITY_POOL_ID`“ durch das `IdentityPoolId` von der Beispielseite des Amazon Cognito Cognito-Identitätspools, das Sie für dieses Beispiel erstellt haben. Dies wird auch an jedes Client-Objekt übergeben.

Rufen Sie die `StartSpeechSynthesisCommand` Methode des Amazon Polly Polly-Client-Serviceobjekts auf, synthetisieren Sie die Sprachnachricht und laden Sie sie in den Amazon S3 S3-Bucket hoch.

```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "../libs/pollyClient.js";

// Create the parameters
var params = {
  OutputFormat: "mp3",
  OutputS3BucketName: "videoanalyzerbucket",
  Text: "Hello David, How are you?",
  TextType: "text",
  VoiceId: "Joanna",
  SampleRate: "22050",
};

const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log("Success, audio file added to " + params.OutputS3BucketName);
  } catch (err) {
    console.log("Error putting object", err);
  }
};

run();
```

Diesen Beispielcode finden Sie [hier auf GitHub](#)

Beispiele für Amazon Redshift

Amazon Redshift ist ein vollständig verwalteter Data-Warehouse-Service in Petabytegröße in der Cloud. Ein Amazon-Redshift-Data-Warehouse ist eine Sammlung von Datenverarbeitungsressourcen, den so genannten Knoten, die zu Gruppen, den so genannten Clustern, zusammengefasst werden. In jedem Cluster wird eine Amazon-Redshift-Engine ausgeführt, und er enthält mindestens eine Datenbank.



Die JavaScript API für Amazon Redshift wird über die [Amazon Redshift Redshift-Client-Klasse](#) verfügbar gemacht.

Themen

- [Beispiele für Amazon Redshift](#)

Beispiele für Amazon Redshift

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Amazon Redshift Redshift-Cluster mit den folgenden Methoden der Redshift Client-Klasse zu erstellen, zu ändern, zu beschreiben und anschließend zu löschen:

- [CreateClusterCommand](#)
- [ModifyClusterCommand](#)
- [DescribeClustersCommand](#)
- [DeleteClusterCommand](#)

Weitere Informationen zu Amazon Redshift-Benutzern finden Sie im [Amazon Redshift Getting Started Guide](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie unter [JavaScript ES6/CommonJS-Syntax](#)

Einen Amazon Redshift Redshift-Cluster erstellen

Dieses Beispiel zeigt, wie Sie mit dem einen Amazon Redshift Redshift-Cluster erstellen. AWS SDK for JavaScript Weitere Informationen finden Sie unter [CreateCluster](#).

Important

Der Cluster, den Sie erstellen möchten, ist aktiv (und läuft nicht in einer Sandbox). Es fallen so lange die standardmäßigen Amazon-Redshift-Nutzungsgebühren für den Cluster an, bis Sie ihn löschen. Wenn Sie den Cluster in derselben Sitzung löschen, in der Sie ihn erstellt haben, sind die Gesamtkosten minimal.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein,

wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie *REGION* durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `redshift-create-cluster.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt, das den Knotentyp, der bereitgestellt werden soll, und die Master-Anmeldedaten für die automatisch im Cluster erstellte Datenbankinstanz und schließlich den Clustertyp angibt.

Note

Ersetzen Sie *CLUSTER_NAME* durch den Namen des Clusters. Geben Sie für *NODE_TYPE* den Knotentyp an, der bereitgestellt werden soll, z. B. „dc2.large“. *MASTER_USERNAME* und *MASTER_USER_PASSWORD* sind die Anmeldeinformationen des Master-Benutzers Ihrer DB-Instance im Cluster. Geben Sie für *CLUSTER_TYPE* den Clustertyp ein. Wenn Sie `single-node` angeben, benötigen Sie den Parameter `NumberOfNodes`. Die übrigen Parameter sind optional.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
  uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
```

```
IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
your cluster needs to access other AWS services on your behalf, such as Amazon S3.
ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
cluster subnet group to be associated with this cluster. Defaults to 'default' if not
specified.
DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-create-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Ändern eines Amazon Redshift Redshift-Clusters

Dieses Beispiel zeigt, wie Sie das Master-Benutzerkennwort eines Amazon Redshift Redshift-Clusters mithilfe von ändern. AWS SDK for JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [ModifyCluster](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
```

```
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `redshift-modify-cluster.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an.

Note

Ersetzen Sie *CLUSTER_NAME* durch den Namen des Clusters und *MASTER_USER_PASSWORD* durch das neue Masterbenutzerkennwort.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-modify-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Details eines Amazon Redshift Redshift-Clusters anzeigen

Dieses Beispiel zeigt, wie Sie die Details eines Amazon Redshift Redshift-Clusters mithilfe von anzeigen. AWS SDK for JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter [DescribeClusters](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `redshift-describe-clusters.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an.

Note

Ersetzen Sie **CLUSTER_NAME** durch den Namen des Clusters.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};
```

```
const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-describe-clusters.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen Sie einen Amazon Redshift Redshift-Cluster

Dieses Beispiel zeigt, wie Sie die Details eines Amazon Redshift Redshift-Clusters mithilfe von anzeigen. AWS SDK for JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [DeleteCluster](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Modul `Node.js` mit dem Namen der Datei `redshift-delete-clusters.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an. Geben Sie dann an, ob Sie vor

dem Löschen einen endgültigen Snapshot des Clusters speichern möchten, und wenn ja, die ID des Snapshots.

Note

Ersetzen Sie *CLUSTER_NAME* durch den Namen des Clusters. Geben Sie für den an *SkipFinalClusterSnapshot*, ob vor dem Löschen ein letzter Snapshot des Clusters erstellt werden soll. Wenn Sie 'false' angeben, geben Sie die ID des endgültigen Cluster-Snapshots in *CLUSTER_SNAPSHOT_ID* an. Sie können diese ID abrufen, indem Sie im Cluster-Dashboard auf den Link in der Spalte Snapshots für den Cluster klicken und nach unten zum Bereich Snapshots scrollen. Beachten Sie, dass der Stamm nicht Teil der Snapshot-ID `rs:` ist.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

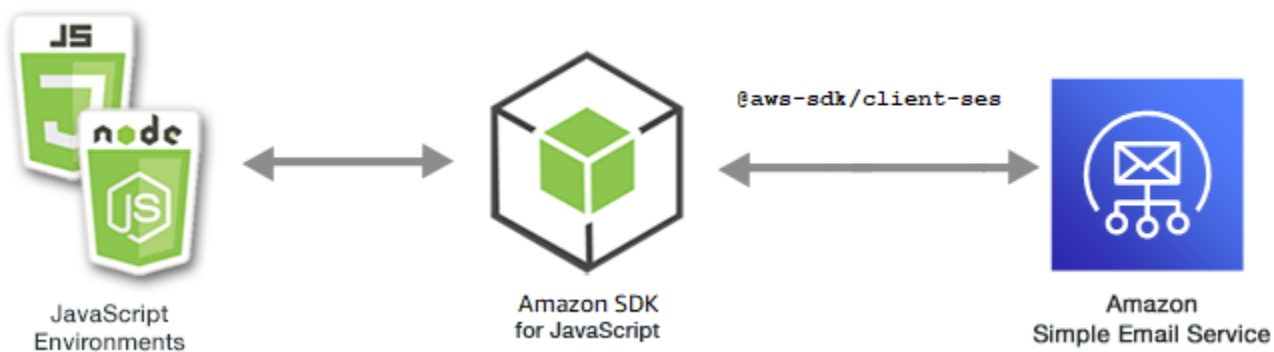
Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node redshift-delete-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Beispiele für Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) ist ein Cloud-basierter E-Mail-Versandservice, der digitale Vermarkter und Anwendungsentwickler beim Versenden von Marketing-, Benachrichtigungs- und Transaktions-E-Mails unterstützt. Dabei handelt es sich um einen zuverlässigen, kosteneffektiven Dienst für Unternehmen jeder Größe, die den Kontakt zu ihren Kunden mithilfe von E-Mail aufrechterhalten



Die JavaScript API für Amazon SES wird über die SES Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Amazon SES SES-Client-Klasse finden Sie unter [Class: SES](#) in der API-Referenz.

Themen

- [Verwaltung von Amazon SES SES-Identitäten](#)
- [Arbeiten mit E-Mail-Vorlagen in Amazon SES](#)
- [Senden von E-Mails mit Amazon SES](#)

Verwaltung von Amazon SES SES-Identitäten



Dieses Node.js-Codebeispiel zeigt:

- So verifizieren Sie E-Mail-Adressen und Domains, die mit Amazon SES verwendet werden.
- So weisen Sie Ihren Amazon SES SES-Identitäten eine AWS Identity and Access Management (IAM-) Richtlinie zu.

- So listen Sie alle Amazon SES SES-Identitäten für Ihr AWS Konto auf.
- So löschen Sie Identitäten, die mit Amazon SES verwendet werden.

Eine Amazon SES-Identität ist eine E-Mail-Adresse oder Domain, die Amazon SES zum Senden von E-Mails verwendet. Amazon SES verlangt von Ihnen, Ihre E-Mail-Identitäten zu verifizieren, um zu bestätigen, dass sie Ihnen gehören, und zu verhindern, dass andere sie verwenden.

Einzelheiten zur Verifizierung von E-Mail-Adressen und Domains in Amazon SES finden Sie unter [Verifizieren von E-Mail-Adressen und Domains in Amazon SES](#) im Amazon Simple Email Service Developer Guide. Informationen zur Sendeautorisierung in Amazon SES finden Sie unter [Überblick über die Amazon SES SES-Sendeautorisierung](#).

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SES SES-Identitäten zu überprüfen und zu verwalten. Die Module Node.js verwenden das SDK JavaScript zur Überprüfung von E-Mail-Adressen und Domains und verwenden dabei die folgenden Methoden der SES Client-Klasse:

- [ListIdentitiesCommand](#)
- [DeleteIdentityCommand](#)
- [VerifyEmailIdentityCommand](#)
- [VerifyDomainIdentityCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

⚠ Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Auflisten Ihrer Identitäten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen und Domains aufzulisten, die mit Amazon SES verwendet werden sollen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listidentities.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den `IdentityType` und andere Parameter für die `ListIdentitiesCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `ListIdentitiesCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben das Parameter-Objekt.

Das `data` zurückgegebene Objekt enthält ein Array von Domänenidentitäten, wie durch den `IdentityType` Parameter angegeben.

Note

IdentityType Ersetzen Sie es durch den Identitätstyp, der "EmailAddress" oder „Domain“ sein kann.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node ses_listidentities.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verifizieren der Identität einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Absender für die Verwendung mit Amazon SES zu verifizieren.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie *REGION* durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
```

```
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifyemailidentity.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich des Herunterladens der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `EmailAddress`-Parameter an die `VerifyEmailIdentityCommand`-Methode der SES-Client-Klasse übergeben. Um die `VerifyEmailIdentityCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

Ersetzen Sie *EMAIL_ADDRESS* durch die E-Mail-Adresse, z. B. `name@example.com`.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifyemailidentity.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Überprüfung einer Domain-Identität

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Domänen für die Verwendung mit Amazon SES zu verifizieren.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifydomainidentity.js`.

Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `Domain`-Parameter an die `VerifyDomainIdentityCommand`-Methode der SES-Client-Klasse übergeben. Um die `VerifyDomainIdentityCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben das Parameter-Objekt.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie *DOMAIN_NAME* durch den Domainnamen.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifydomainidentity.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Identitäten löschen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen oder Domains zu löschen, die mit Amazon SES verwendet werden.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deleteidentity.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `Identity`-Parameter an die `DeleteIdentityCommand`-Methode der SES-Client-Klasse übergeben. Um die `DeleteIdentityCommand` Methode aufzurufen, erstellen Sie ein Serviceobjekt `request` für den Aufruf eines Amazon SES SES-Clients und übergeben dabei die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie **IDENTITY_EMAIL** durch die E-Mail-Adresse der Identität, die gelöscht werden soll.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node ses_deleteidentity.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Arbeiten mit E-Mail-Vorlagen in Amazon SES



Dieses Node.js-Codebeispiel zeigt:

- So erhalten Sie eine Liste all Ihrer E-Mail-Vorlagen.
- So rufen Sie E-Mail-Vorlagen ab und aktualisieren sie.
- So erstellen und löschen Sie E-Mail-Vorlagen.

Amazon SES ermöglicht es Ihnen, personalisierte E-Mail-Nachrichten mithilfe von E-Mail-Vorlagen zu versenden. Einzelheiten zum Erstellen und Verwenden von E-Mail-Vorlagen in Amazon SES finden Sie unter [Senden personalisierter E-Mails mit der Amazon SES SES-API](#) im Amazon Simple Email Service Developer Guide.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js-Module, um mit E-Mail-Vorlagen zu arbeiten. Die Module von Node.js verwenden das SDK JavaScript, um E-Mail-Vorlagen mithilfe der folgenden Methoden der SES Client-Klasse zu erstellen und zu verwenden:

- [ListTemplatesCommand](#)
- [CreateTemplateCommand](#)
- [GetTemplateCommand](#)
- [DeleteTemplateCommand](#)
- [UpdateTemplateCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Ihre E-Mail-Vorlagen auflisten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listtemplates.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, mit dem Sie die Parameter für die `ListTemplatesCommand`-Methode der SES-Client-Klasse übergeben können. Um die `ListTemplatesCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
```

```
const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein. Amazon SES gibt die Liste der Vorlagen zurück.

```
node ses_listtemplates.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Eine E-Mail-Vorlage erhalten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_gettemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `TemplateName`-Parameter an die `GetTemplateCommand`-Methode der `SES-Client`-Klasse übergeben. Um die `GetTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES `SES-Client-Service`objekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie `TEMPLATE_NAME` durch den Namen der Vorlage, die zurückgegeben werden soll.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
}
```

```
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_gettemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Eine E-Mail-Vorlage erstellen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createtemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameter für die `CreateTemplateCommand`-Methode der SES-Client-Klasse zu übergeben, einschließlich `TemplateName`, `HtmlPart`, `SubjectPart` und `TextPart`. Um die `CreateTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie

können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie *TEMPLATE_NAME* durch einen Namen für die neue Vorlage, *HtmlPart* durch den HTML-markierten Inhalt der E-Mail und *SubjectPart* durch den Betreff der E-Mail.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
```

```
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein. Die Vorlage wird zu Amazon SES hinzugefügt.

```
node ses_createtemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Aktualisieren einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_updatetemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die `Template-Parameterwerte`, die Sie in der Vorlage aktualisieren möchten, mit dem erforderlichen `TemplateName-Parameter` an die `updateTemplateCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `updateTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

Dieses Beispiel importiert und verwendet die erforderlichen AWS Service V3-Paketclients und V3-Befehle und verwendet die `send` Methode in einem Async/Await-Muster. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie `TEMPLATE_NAME` durch den Namen der Vorlage und `HTML_PART` durch den HTML-markierten Inhalt der E-Mail.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
}
```

```
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_updatetemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Löschen einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).


Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletetemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den erforderlichen `TemplateName`-Parameter an die `DeleteTemplateCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `DeleteTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

Dieses Beispiel importiert und verwendet die erforderlichen AWS Service V3-Paketclients und V3-Befehle und verwendet die `send` Methode in einem `Async/Await`-Muster. Sie

können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

 Note

Ersetzen Sie *TEMPLATE_NAME* durch den Namen der Vorlage, die gelöscht werden soll.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_deletetemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Senden von E-Mails mit Amazon SES



Dieses Node.js-Codebeispiel zeigt:

- das Senden einer Text- oder HTML-E-Mail.
- das Senden von E-Mails, die auf einer E-Mail-Vorlage basieren.
- das Senden von Massen-E-Mails, die auf einer E-Mail-Vorlage basieren.

Die Amazon SES SES-API bietet Ihnen zwei verschiedene Möglichkeiten, eine E-Mail zu senden, je nachdem, wie viel Kontrolle Sie über die Zusammensetzung der E-Mail-Nachricht haben möchten: formatiert und roh. Weitere Informationen finden Sie unter [Senden formatierter E-Mails mit der Amazon SES SES-API](#) und [Senden von Roh-E-Mails mit der Amazon SES SES-API](#).

Das Szenario

In diesem Beispiel verwenden Sie mehrere Node.js-Module, um E-Mails auf verschiedene Weisen zu senden. Die Module von Node.js verwenden das SDK JavaScript, um E-Mail-Vorlagen mithilfe der folgenden Methoden der SES Client-Klasse zu erstellen und zu verwenden:

- [SendEmailCommand](#)
- [SendTemplatedEmailCommand](#)
- [SendBulkTemplatedEmailCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen

finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Anforderungen für den Versand von E-Mail-Nachrichten

Amazon SES verfasst eine E-Mail-Nachricht und stellt sie sofort in die Warteschlange für den Versand. Damit Sie mithilfe der `SendEmailCommand`-Methode E-Mails senden können, muss Ihre Nachricht die folgenden Anforderungen erfüllen:

- Sie müssen die Nachricht von einer verifizierten E-Mail-Adresse oder Domäne senden. Wenn Sie versuchen, E-Mails über eine nicht verifizierte Adresse oder Domäne zu senden, führt die Operation zu einem "Email address not verified"-Fehler.
- Wenn sich Ihr Konto noch in der Amazon SES Sandbox befindet, können Sie die E-Mails nur an verifizierte Adressen oder Domänen oder E-Mail-Adressen des Amazon SES-Postfachsimulators senden. Weitere Informationen finden Sie unter [Verifizieren von E-Mail-Adressen und Domains](#) im Amazon Simple Email Service Developer Guide.
- Die Gesamtgröße der Nachricht, einschließlich Anlagen, muss kleiner als 10 MB sein.
- Die Nachricht muss mindestens eine E-Mail-Adresse für einen Empfänger enthalten. Bei der Empfänger-Adresse kann es sich um eine Empfängeradresse, eine CC: Adresse oder BCC: Adresse handeln. Wenn die E-Mail-Adresse eines Empfängers nicht gültig ist (d. h., sie hat nicht das Format `UserName@[SubDomain.]Domain.TopLevelDomain`), wird die gesamte Nachricht zurückgewiesen, auch wenn die Nachricht andere gültige Empfänger enthält.
- Die Nachricht darf in den Feldern An:, CC: und BCC: nicht mehr als 50 Empfänger enthalten. Wenn Sie eine E-Mail an eine größere Zielgruppe senden möchten, können Sie Ihre Empfängerliste

in Gruppen von höchstens 50 unterteilen und dann die `sendEmail`-Methode aufrufen, um die Nachricht mehrmals an die einzelnen Gruppen zu senden.

Eine E-Mail senden

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, einschließlich Absender- und Empfängeradressen, Betreff und E-Mail-Text in Klartext- und HTML-Formaten, an die `SendEmailCommand` Methode der SES Client-Klasse zu übergeben. Um die `SendEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

Dieses Beispiel importiert und verwendet die erforderlichen AWS Service V3-Paketclients und V3-Befehle und verwendet die `send` Methode in einem `Async/Await`-Muster. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie *toAddress* durch die Adresse, an die die E-Mail gesendet werden soll, und *fromAddress* durch die E-Mail-Adresse, von der die E-Mail gesendet werden soll.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ]
  });
}
```

```
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendemail.js
```

Diesen Beispielcode [finden Sie hier auf GitHub](#).

Senden einer E-Mail mithilfe einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendtemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, z. B. die Sender- und Empfängeradressen, Betreff, E-Mail-Text im Klartext- und HTML-Format, an die `SendTemplatedEmailCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `SendTemplatedEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie *REGION* durch Ihre AWS Region, *USER* durch den Namen und die E-Mail-Adresse, an die die E-Mail gesendet werden soll, *VERIFIED_EMAIL* durch die E-Mail-Adresse, von der die E-Mail gesendet werden soll, und *TEMPLATE_NAME* durch den *Namen* der Vorlage.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
```

```
*/
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendtemplatedemail.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Senden von Massen-E-Mails mithilfe einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES `SES-Client`-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `ses_sendbulktemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, einschließlich Absender- und Empfängeradressen, Betreff und E-Mail-Text in Klartext- und HTML-Formaten, an die `SendBulkTemplatedEmailCommand` Methode der `SES Client`-Klasse zu übergeben. Um die `SendBulkTemplatedEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES `SES-Serviceobjekt` auf und übergeben die Parameter.

Note

Dieses Beispiel importiert und verwendet die erforderlichen AWS Service V3-Paketclients und V3-Befehle und verwendet die `send` Methode in einem `Async/Await`-Muster. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

Ersetzen Sie **USERS** durch die Namen und E-Mail-Adressen, an die die E-Mail gesendet werden soll, **VERIFIED_EMAIL_1** durch die E-Mail-Adresse, von der die E-Mail gesendet werden soll, und **TEMPLATE_NAME** durch den Namen der Vorlage.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
```

```
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
     user
     * to a 'Destination' and provide user specific replacement data to create
     personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
  });
};
```

```
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Beispiele für Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) ist ein Webservice, der die Zustellung oder das Senden von Nachrichten an abonnierende Endpunkte oder Clients koordiniert und verwaltet.

In Amazon SNS gibt es zwei Arten von Kunden — Herausgeber und Abonnenten —, die auch als Produzenten und Verbraucher bezeichnet werden.



Herausgeber kommunizieren asynchron mit Abonnenten, indem sie eine Nachricht erstellen und an ein Thema senden, bei dem es sich wirklich um einen logischen Zugriffspunkt und Kommunikationskanal handelt. Abonnenten (Webserver, E-Mail-Adressen, Amazon SQS-Warteschlangen, AWS Lambda Funktionen) konsumieren oder empfangen die Nachricht oder Benachrichtigung über eines der unterstützten Protokolle (Amazon SQS, HTTP/S, E-Mail, SMS/AWS Lambda), wenn sie das Thema abonniert haben.

Die JavaScript API für Amazon SNS wird über die [Klasse: SNS](#) verfügbar gemacht.

Themen

- [Themen in Amazon SNS verwalten](#)
- [Nachrichten in Amazon SNS veröffentlichen](#)
- [Verwaltung von Abonnements in Amazon SNS](#)
- [Senden von SMS-Nachrichten mit Amazon SNS](#)

Themen in Amazon SNS verwalten



Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie Themen in Amazon SNS, zu denen Sie Benachrichtigungen veröffentlichen können.
- So löschen Sie in Amazon SNS erstellte Themen.
- Abrufen einer Liste von verfügbaren Themen

- Abrufen und Festlegen von Themenattributen

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SNS SNS-Themen zu erstellen, aufzulisten und zu löschen und Themenattribute zu verarbeiten. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der SNS Client-Klasse:

- [CreateTopicCommand](#)
- [ListTopicsCommand](#)
- [DeleteTopicCommand](#)
- [GetTopicAttributesCommand](#)
- [SetTopicAttributesCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Erstellen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie `REGION` durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `create-topic.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den Namen (Name) für das neue Thema an die `CreateTopicCommand`-Methode der SNS-Client-Klasse zu übergeben. Um die `CreateTopicCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt. Die `data` zurückgegebene enthält den ARN des Themas.

Note

Ersetzen Sie `TOPIC_NAME` durch den Namen des Themas.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
```

```
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node create-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Auflisten Ihrer -Themen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Amazon SNS SNS-Themen aufzulisten.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-topics.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein leeres Objekt, das an die `ListTopicsCommand`-Methode der `SNS-Client`-Klasse übergeben wird. Um die `ListTopicsCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt. Das data zurückgegebene Objekt enthält ein Array mit Ihrem Thema Amazon Resource Names (ARNs).

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-topics.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu löschen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.


```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `delete-topic.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, das den `TopicArn` des zu löschenden Themas enthält, um es an die `DeleteTopicCommand`-Methode der SNS-Client-Klasse zu übergeben. Um die `DeleteTopicCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie `TOPIC_ARN` durch den Amazon-Ressourcennamen (ARN) des Themas, das Sie löschen möchten.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
// totalRetryDelay: 0
// }
// }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node delete-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abrufen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um Attribute eines Amazon SNS SNS-Themas abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get-topic-attributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn` eines zu löschenden Themas enthält, um es an die `GetTopicAttributesCommand`-Methode der SNS-Client-Klasse zu übergeben. Um die `GetTopicAttributesCommand` Methode aufzurufen, wird ein Amazon SNS SNS-Client-Serviceobjekt aufgerufen und das Parameter-Objekt übergeben.

Note

Ersetzen Sie **TOPIC_ARN** durch den ARN des Themas.

```

import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};

```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node get-topic-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Festlegen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die veränderbaren Attribute eines Amazon SNS SNS-Themas festzulegen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `set-topic-attributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für eine Aktualisierung des Attributs enthält. Dazu gehören der TopicArn des Themas, dessen Attribute Sie festlegen möchten, der Name des festzulegenden Attributs und der neue Wert für dieses Attribut. Sie können nur die Attribute `Policy`, `DisplayName` und `DeliveryPolicy` festlegen. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse SNS. Um die `SetTopicAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie **ATTRIBUTE_NAME** durch den Namen des Attributs, das Sie festlegen, **TOPIC_ARN** durch den Amazon-Ressourcennamen (ARN) des Themas, dessen Attribute Sie festlegen möchten, und **NEW_ATTRIBUTE_VALUE** durch den neuen Wert für dieses Attribut.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node set-topic-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Nachrichten in Amazon SNS veröffentlichen



Dieses Node.js-Codebeispiel zeigt:

- So veröffentlichen Sie Nachrichten zu einem Amazon SNS SNS-Thema.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Nachrichten von Amazon SNS an Themenendpunkte, E-Mails oder Telefonnummern zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript , um Nachrichten mit dieser Methode der SNS Client-Klasse zu senden:

- [PublishCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Veröffentlichen einer Nachricht in einem SNS-Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine Nachricht zu einem Amazon SNS SNS-Thema zu veröffentlichen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS `SNS-Client`-Objekt erstellt wird. Ersetzen Sie *REGION* durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `publish-topic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für die Veröffentlichung einer Nachricht enthält, einschließlich des Nachrichtentextes und des Amazon-Ressourcennamens (ARN) des Amazon `SnsTopic`. Weitere Details zu verfügbaren SMS-Attributen finden Sie unter [SetSMSAttributes](#).

Übergeben Sie die Parameter an die `PublishCommand` Methode der `SNS Client`-Klasse. Erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Service`objekt aufruft und das Parameterobjekt übergibt.

Note

Ersetzen Sie *MESSAGE_TEXT* durch den Nachrichtentext und *TOPIC_ARN* durch den ARN des SNS-Themas.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 * string or an object
 *
 *                                     if you are using the `json`
 * `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
```

```
message = "Hello from SNS!",
topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node publish-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verwaltung von Abonnements in Amazon SNS



Dieses Node.js-Codebeispiel zeigt:

- So listen Sie alle Abonnements für ein Amazon SNS SNS-Thema auf.
- So abonnieren Sie eine E-Mail-Adresse, einen Anwendungsendpunkt oder eine AWS Lambda Funktion für ein Amazon SNS SNS-Thema.

- So melden Sie sich von Amazon SNS SNS-Themen ab.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Benachrichtigungen zu Amazon SNS SNS-Themen zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der SNS Client-Klasse:

- [ListSubscriptionsByTopicCommand](#)
- [SubscribeCommand](#)
- [ConfirmSubscriptionCommand](#)
- [UnsubscribeCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Auflisten von Abonnements eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Abonnements für ein Amazon SNS SNS-Thema aufzulisten.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-subscriptions-by-topic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn`-Parameter für das Thema enthält, dessen Abonnements Sie auflisten möchten. Übergeben Sie die Parameter an die `ListSubscriptionsByTopicCommand`-Methode der Client-Klasse SNS. Um die `ListSubscriptionsByTopicCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie **TOPIC_ARN** durch den Amazon-Ressourcennamen (ARN) für das Thema, dessen Abonnements Sie auflisten möchten.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 * subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
```

```
const response = await snsClient.send(
  new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
);

console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   Subscriptions: [
//     {
//       SubscriptionArn: 'PendingConfirmation',
//       Owner: '901487484989',
//       Protocol: 'email',
//       Endpoint: 'corepyle@amazon.com',
//       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
//     }
//   ]
// }
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-subscriptions-by-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren eines Themas durch Hinterlegen einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Adresse zu abonnieren, sodass sie SMTP-E-Mail-Nachrichten von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
  blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-email.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol`-Parameter enthält, um das `email`-Protokoll, den `TopicArn` für das Thema, das abonniert werden soll, und eine E-Mail-Adresse als Endpoint der Nachricht anzugeben. Übergeben Sie die Parameter an die `SubscribeCommand`-Methode der Client-Klasse `SNS`. Sie können die `subscribe` Methode verwenden, um mehrere verschiedene Endpunkte für ein Amazon SNS SNS-Thema zu abonnieren, abhängig von den Werten, die für die übergebenen Parameter verwendet werden, wie andere Beispiele in diesem Thema zeigen werden.

Um die `SubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie *TOPIC_ARN* durch den Amazon-Ressourcennamen (ARN) für das Thema und *EMAIL_ADDRESS* durch die E-Mail-Adresse, die Sie abonnieren möchten.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
  subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "user@me.com",
) => {
  const response = await snsClient.send(
```

```
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node subscribe-email.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Bestätigen von Abonnements

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die Absicht eines Endpunktbesitzers zu überprüfen, E-Mails zu empfangen, indem Sie das Token validieren, das durch eine vorherige Abonnement-Aktion an den Endpunkt gesendet wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

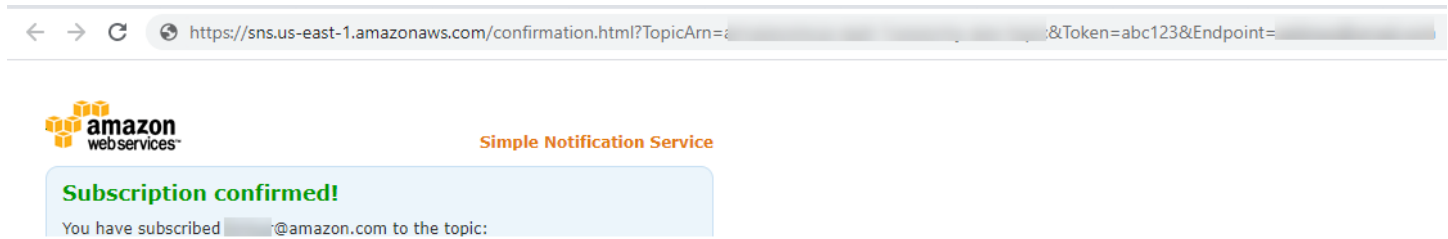
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `confirm-subscription.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Definieren Sie die Parameter, einschließlich `TOPIC_ARN` und `TOKEN`, und definieren Sie einen Wert von `TRUE` oder `FALSE` für `AuthenticateOnUnsubscribe`.

Das Token ist ein kurzlebiges Token, das während einer vorherigen `SUBSCRIBE` Aktion an den Besitzer eines Endpunkts gesendet wurde. Bei einem E-Mail-Endpunkt `TOKEN` steht es beispielsweise in der URL der E-Mail zur Bestätigung des Abonnements, die an den E-Mail-Besitzer gesendet wurde. Zum Beispiel `abc123` ist das Token in der folgenden URL.



Um die `ConfirmSubscriptionCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie `TOPIC_ARN` durch den Amazon-Ressourcennamen (ARN) für das Thema, `TOKEN` durch den Token-Wert aus der URL, die in einer früheren `Subscribe` Aktion an den Endpoint-Besitzer gesendet wurde, und definiere `AuthenticateOnUnsubscribe` mit dem Wert oder `TRUE` `FALSE`

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                        that are not AWS services (HTTP/S, email) need to be
 *                        confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 *                            subscription.
 */
export const confirmSubscription = async (
```

```
token = "TOKEN",
topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
  xxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node confirm-subscription.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren eines Themas mit einem Anwendungsendpunkt

Verwenden Sie in diesem Beispiel ein Modul Node.js, um einen mobilen Anwendungsendpunkt zu abonnieren, sodass dieser Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-app.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Module und Pakete.

Erstellen Sie ein Objekt, das den `Protocol` Parameter zur Angabe des `application` Protokolls, das `TopicArn` für das zu abonnierende Thema und den Amazon-Ressourcennamen (ARN) eines mobilen Anwendungsendpunkts für den `Endpoint` Parameter enthält. Übergeben Sie die Parameter an die `SubscribeCommand`-Methode der `Client`-Klasse `SNS`.

Um die `SubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt.

Note

Ersetzen Sie *TOPIC_ARN* durch den Amazon-Ressourcennamen (ARN) für das Thema und *MOBILE_ENDPOINT_ARN* durch den Endpunkt, den Sie für das Thema abonnieren.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 * created
 *
 *                               when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
```



```
    Protocol: "application",
    TopicArn: topicArn,
    Endpoint: endpoint,
  )),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node subscribe-app.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren einer Lambda-Funktion für ein Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine AWS Lambda Funktion zu abonnieren, sodass sie Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-lambda.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol` Parameter enthält und `lambda` das Protokoll, das `TopicArn` für das zu abonnierende Thema und den Amazon-Ressourcennamen (ARN) einer AWS Lambda Funktion als `Endpoint` Parameter angibt. Übergeben Sie die Parameter an die `SubscribeCommand`-Methode der Client-Klasse `SNS`.

Um die `SubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie *TOPIC_ARN* durch den Amazon-Ressourcennamen (ARN) für das Thema und *LAMBDA_FUNCTION_ARN* durch den Amazon-Ressourcennamen (ARN) der Lambda-Funktion.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
```

```
//    httpStatusCode: 200,  
//    requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',  
//    extendedRequestId: undefined,  
//    cfId: undefined,  
//    attempts: 1,  
//    totalRetryDelay: 0  
//  },  
//  SubscriptionArn: 'pending confirmation'  
// }  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node subscribe-lambda.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abmelden von einem Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Themenabonnement zu kündigen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `unsubscribe.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, das den `SubscriptionArn` Parameter enthält und den Amazon-Ressourcennamen (ARN) des Abonnements angibt, das gekündigt werden soll. Übergeben Sie die Parameter an die `UnsubscribeCommand`-Methode der Client-Klasse SNS.

Um die `UnsubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das `Parameter-Objekt` übergibt.

Note

Ersetzen Sie `TOPIC_SUBSCRIPTION_ARN` durch den Amazon-Ressourcennamen (ARN) des Abonnements, um das Abonnement zu kündigen.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node unsubscribe.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Senden von SMS-Nachrichten mit Amazon SNS



Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie SMS-Nachrichteneinstellungen für Amazon SNS ab und legen sie fest.
- Überprüfen, ob eine Telefonnummer vom Empfang von SMS-Nachrichten abgemeldet wurde
- Abrufen einer Liste der Telefonnummern, die vom Empfang von SMS-Nachrichten abgemeldet wurden
- Senden einer SMS-Nachricht

Das Szenario

Verwenden Sie Amazon SNS, um Textnachrichten oder SMS-Nachrichten an SMS-fähige Geräte zu senden. Sie können eine Nachricht direkt an eine Telefonnummer senden oder Sie können eine Nachricht an mehrere Telefonnummern gleichzeitig senden, indem Sie das Thema für diese Telefonnummern abonnieren und die Nachricht an das Thema senden.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um SMS-Textnachrichten von Amazon SNS auf SMS-fähigen Geräten zu veröffentlichen. Die Module Node.js verwenden das SDK für JavaScript die Veröffentlichung von SMS-Nachrichten mit den folgenden Methoden der SNS Client-Klasse:

- [GetSMSAttributesCommand](#)
- [SetSMSAttributesCommand](#)
- [CheckIfPhoneNumberIsOptedOutCommand](#)
- [ListPhoneNumbersOptedOutCommand](#)
- [PublishCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#).

Abrufen von SMS-Attributen

Verwenden Sie Amazon SNS, um Einstellungen für SMS-Nachrichten festzulegen, z. B. wie Ihre Lieferungen optimiert werden (aus Kostengründen oder für eine zuverlässige Zustellung), Ihr monatliches Ausgabenlimit, wie Nachrichtenzustellungen protokolliert werden und ob Sie tägliche SMS-Nutzungsberichte abonnieren möchten. Diese Einstellungen werden abgerufen und als SMS-Attribute für Amazon SNS festgelegt.

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get-sms-attributes.js`.

Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter zum Abrufen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die abgerufen werden. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter [SetSMSAttributes](#) in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut abgerufen. Dieses Attribut steuert, ob SMS-Nachrichten als `Promotional` oder als `Transactional` gesendet werden. Im ersten Fall wird die Nachrichtenzustellung im Hinblick auf die Kosten und im zweiten Fall im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse `SNS`. Um die `SetSMSAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie *ATTRIBUTE_NAME* durch den Namen des Attributs.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
//     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   attributes: { DefaultSMSType: 'Transactional' }
// }
return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node get-sms-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Festlegen von SMS-Attributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `set-sms-attribute-type.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter zum Festlegen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die festgelegt werden, und der jeweils festzulegenden Werte. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter [SetSMSAttributes](#) in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut auf `Transactional` festgelegt. Damit wird die Nachrichtenzustellung im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse SNS. Um die `SetSMSAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node set-sms-attribute-type.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Überprüfen, ob für eine Telefonnummer der Empfang deaktiviert wurde

In diesem Beispiel verwenden Sie ein Node.js-Modul, um zu überprüfen, ob für eine Telefonnummer der Empfang von SMS-Nachrichten deaktiviert wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `check-if-phone-number-is-opted-out.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die zu überprüfende Telefonnummer als Parameter enthält.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer anzugeben, die überprüft werden soll. Übergeben Sie das Objekt an die `CheckIfPhoneNumberIsOptedOutCommand`-Methode der Client-Klasse `SNS`. Um die `CheckIfPhoneNumberIsOptedOutCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

1.

Ersetzen Sie **PHONE_NUMBER** durch die Telefonnummer.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";
```

```
export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node check-if-phone-number-is-opted-out.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Auflisten deaktivierter Telefonnummern

In diesem Beispiel verwenden Sie ein Node.js-Modul, um eine Liste der Telefonnummern abzurufen, für die der Empfang von SMS-Nachrichten deaktiviert wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-phone-numbers-opted-out.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein leeres Objekt als Parameter.

Übergeben Sie das Objekt an die `ListPhoneNumbersOptedOutCommand`-Methode der Client-Klasse `SNS`. Um die `ListPhoneNumbersOptedOutCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Service`objekt aufruft und das Parameter-Objekt übergibt.

```
import { ListPhoneNumbersOptedOutCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listPhoneNumbersOptedOut = async () => {
  const response = await snsClient.send(
    new ListPhoneNumbersOptedOutCommand({}),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   phoneNumbers: ['+15555550100']
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-phone-numbers-opted-out.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Veröffentlichen einer SMS-Nachricht

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden einer SMS-Nachricht an eine Telefonnummer.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `publish-sms.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter `Message` und `PhoneNumber` enthält.

Wenn Sie eine SMS-Nachricht senden, geben Sie die Telefonnummer im E.164-Format an. Die Richtlinie E.164 legt die internationale Schreibweise für Telefonnummern fest. Rufnummern im E.164-Format bestehen aus maximal 15 Zeichen sowie einem vorangestellten Plus-Zeichen (+) und der Ländervorwahl. Eine US-Telefonnummer im E.164-Format sieht beispielsweise wie folgt aus: `+1001XXX5550100`.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer zum Senden der Nachricht anzugeben. Übergeben Sie das Objekt an die `PublishCommand`-Methode der Client-Klasse `SNS`. Um die `PublishCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt.

Note

Ersetzen Sie **TEXT_MESSAGE** durch die *Textnachricht* und **PHONE_NUMBER** durch die Telefonnummer.

```
import { PublishCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *
 *                                     if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxx'
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node publish-sms.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Beispiele für Amazon Transcribe

Amazon Transcribe macht es Entwicklern leicht, ihren Anwendungen Sprache-zu-Text-Funktionen hinzuzufügen.



Die JavaScript API für Amazon Transcribe wird über die [TranscribeServiceClient](#)-Klasse verfügbar gemacht.

Themen

- [Beispiele für Amazon Transcribe](#)
- [Medizinische Beispiele von Amazon Transcribe](#)

Beispiele für Amazon Transcribe

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Transkriptionsaufträge mit den folgenden Methoden der Client-Klasse zu erstellen, aufzulisten und zu löschen:

TranscribeService

- [StartTranscriptionJobCommand](#)
- [ListTranscriptionJobsCommand](#)
- [DeleteTranscriptionJobCommand](#)

Weitere Informationen zu Amazon Transcribe Transcribe-Benutzern finden Sie im Amazon Transcribe [Transcribe-Entwicklerhandbuch](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie unter [JavaScript ES6/CommonJS-Syntax](#)

Einen Amazon Transcribe Transcribe-Job starten

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem starten. AWS SDK for JavaScript Weitere Informationen finden Sie unter [StartTranscriptionJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-create-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt und geben Sie die erforderlichen Parameter an. Starten Sie den Job mit dem `StartMedicalTranscriptionJobCommand` Befehl.

Note

Ersetzen Sie *MEDICAL_JOB_NAME* durch *einen Namen* für den Transkriptionsjob. Geben Sie für *OUTPUT_BUCKET_NAME* den Amazon S3 S3-Bucket an, in dem die Ausgabe gespeichert wird. Geben Sie für *JOB_TYPE* die *Auftragstypen* an. Geben Sie für *SOURCE_LOCATION* den Speicherort der Quelldatei an. Geben Sie für *SOURCE_FILE_LOCATION* den *Speicherort* der Eingabemediendatei an.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "./libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-create-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Amazon Transcribe Transcribe-Jobs auflisten

Dieses Beispiel zeigt, wie die Amazon Transcribe-Transkriptionsaufträge mithilfe von aufgelistet werden. AWS SDK for JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [ListTranscriptionJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-list-jobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern.

Note

Ersetzen Sie **KEY_WORD** durch ein Schlüsselwort, das der zurückgegebene Jobname enthalten muss.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-list-jobs.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines Amazon Transcribe Transcribe-Jobs

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem löschen. AWS SDK for JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter.

[DeleteTranscriptionJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-delete-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region und den Namen des Jobs an, den Sie löschen möchten.

Note

Ersetzen Sie *JOB_NAME* durch den Namen des zu löschenden Jobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-delete-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Medizinische Beispiele von Amazon Transcribe

In diesem Beispiel werden eine Reihe von Modulen von Node.js verwendet, um medizinische Transkriptionsaufträge mit den folgenden Methoden der Client-Klasse zu erstellen, aufzulisten und zu löschen: `TranscribeService`

- [StartMedicalTranscriptionJobCommand](#)
- [ListMedicalTranscriptionJobsCommand](#)
- [DeleteMedicalTranscriptionJobCommand](#)

Weitere Informationen zu Amazon Transcribe Transcribe-Benutzern finden Sie im [Amazon Transcribe Transcribe-Entwicklerhandbuch](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie unter [JavaScript ES6/CommonJS-Syntax](#)

Starten eines medizinischen Transkriptionsauftrags von Amazon Transcribe

Dieses Beispiel zeigt, wie Sie einen medizinischen Transkriptionsauftrag von Amazon Transcribe mit dem `start` starten. AWS SDK for JavaScript Weitere Informationen finden Sie unter [MedicalTranscriptionJob starten](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-create-medical-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt und geben Sie die erforderlichen Parameter an. Starten Sie den medizinischen Job mit dem `StartMedicalTranscriptionJobCommand` Befehl.

Note

Ersetzen Sie **MEDICAL_JOB_NAME** durch *einen Namen* für den medizinischen Transkriptionsjob. Geben Sie für **OUTPUT_BUCKET_NAME** den Amazon S3 S3-Bucket an, in dem die Ausgabe gespeichert wird. Geben Sie für **JOB_TYPE** die *Auftragstypen* an. Geben Sie für **SOURCE_LOCATION** den Speicherort der Quelldatei an. Geben Sie für **SOURCE_FILE_LOCATION** den *Speicherort* der Eingabemediendatei an.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    // region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-create-medical-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Stellenangebote von Amazon Transcribe im Gesundheitswesen auflisten

Dieses Beispiel zeigt, wie Sie die Amazon Transcribe-Transkriptionsaufträge mithilfe von auflisten. AWS SDK for JavaScript [Weitere Informationen finden Sie unter Befehl. ListTranscriptionMedicalJobs](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie `REGION` durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-list-medical-jobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern und listen Sie die medizinischen Aufgaben mithilfe des `ListMedicalTranscriptionJobsCommand` Befehls auf.

Note

Ersetzen Sie `KEYWORD` durch ein Schlüsselwort, das der zurückgegebene Jobname enthalten muss.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
  string
};
```



```
export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params)
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-list-medical-jobs.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines medizinischen Jobs bei Amazon Transcribe

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem löschen. AWS SDK for JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter.

[DeleteTranscriptionMedicalJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie **REGION** durch Ihre AWS Region.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-delete-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation

der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern und löschen Sie den medizinischen Job mithilfe des `DeleteMedicalJobCommand` Befehls.

Note

Ersetzen Sie *JOB_NAME* durch den Namen des zu löschenden Jobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-delete-medical-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Node.js auf einer Amazon EC2 EC2-Instance einrichten

Ein gängiges Szenario für die Verwendung von Node.js mit dem SDK für JavaScript ist die Einrichtung und Ausführung einer Node.js -Webanwendung auf einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance. In diesem Tutorial erstellen Sie eine Linux-Instance, stellen eine Verbindung zur Instance über SSH her und installieren anschließend Node.js, um es auf dieser Instance auszuführen.

Voraussetzungen

In diesem Tutorial wird davon ausgegangen, dass Sie bereits eine Linux-Instance mit einem öffentlichen DNS-Namen gestartet haben, der über das Internet erreichbar ist und zu der Sie über SSH eine Verbindung herstellen können. Weitere Informationen finden Sie unter [Schritt 1: Starten einer Instance](#) im Benutzerhandbuch zu Amazon EC2.

Important

Verwenden Sie das Amazon Linux 2023 Amazon Machine Image (AMI), wenn Sie eine neue Amazon EC2 EC2-Instance starten.

Außerdem müssen Sie Ihre Sicherheitsgruppe so konfiguriert haben, dass Verbindungen über SSH (Port 22), HTTP (Port 80) und HTTPS (Port 443) erlaubt sind. Weitere Informationen zu diesen Voraussetzungen finden Sie unter [Einrichtung mit Amazon EC2](#) im Amazon EC2 EC2-Benutzerhandbuch.

Verfahren

Mithilfe des folgenden Verfahrens können Sie Node.js auf einer Amazon Linux-Instance installieren. Sie können diesen Server zum Hosten einer Node.js-Webanwendung verwenden.

So richten Sie Node.js auf Ihrer Linux-Instance ein

1. Stellen Sie als `ec2-user` eine Verbindung mit Ihrer Linux-Instance über SSH her.
2. Installieren Sie den Node Version Manager (`nvm`), indem Sie in der Befehlszeile Folgendes eingeben.

⚠ Warning

AWS steuert den folgenden Code nicht. Bevor Sie ihn ausführen, überprüfen Sie unbedingt dessen Authentizität und Integrität. Weitere Informationen zu diesem Code finden Sie im [GitHubnvm-Repository](#).

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Wir werden Node.js verwenden, da mehrere Versionen von Node.js installiert werden können und Sie zwischen ihnen wechseln können.

3. Laden Sie, nvm indem Sie in der Befehlszeile Folgendes eingeben.

```
source ~/.bashrc
```

4. Verwenden Sie nvm, um die neueste LTS-Version von Node.js zu installieren, indem Sie in der Befehlszeile Folgendes eingeben.

```
nvm install --lts
```

Bei der Installation von Node.js wird auch der Node Package Manager (npm) installiert, sodass Sie bei Bedarf zusätzliche Module installieren können.

5. Testen Sie, ob Node.js installiert ist und ordnungsgemäß ausgeführt wird. Geben Sie dazu den folgenden Befehl in die Befehlszeile ein.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Dadurch erscheint folgende Meldung, in der die ausgeführte Node.js-Version angezeigt wird.

Running Node.js *VERSION*

ℹ Note

Die Knoteninstallation gilt nur für die aktuelle Amazon EC2-Sitzung. Wenn Sie Ihre CLI-Sitzung neu starten, müssen Sie nvm erneut verwenden, um die installierte Knotenversion zu aktivieren. Wenn die Instance beendet wird, müssen Sie den Knoten erneut installieren. Die

Alternative besteht darin, ein Amazon Machine Image (AMI) der Amazon EC2 EC2-Instance zu erstellen, sobald Sie die Konfiguration haben, die Sie behalten möchten, wie im folgenden Thema beschrieben.

Ein Amazon Machine Image (AMI) erstellen

Nachdem Sie Node.js auf einer Amazon EC2 EC2-Instance installiert haben, können Sie aus dieser Instance ein Amazon Machine Image (AMI) erstellen. Die Erstellung eines AMI macht es einfach, mehrere Amazon EC2 EC2-Instances mit derselben Node.js Installation bereitzustellen. Weitere Informationen zum Erstellen eines AMI aus einer vorhandenen Instance finden Sie unter [Creating an amazon EBS-backed Linux AMI](#) im Amazon EC2 EC2-Benutzerhandbuch.

Zugehörige Ressourcen

Weitere Informationen zu den in diesem Thema verwendeten Befehlen und der Software finden Sie auf den folgenden Webseiten:

- Node Version Manager (nvm) — Siehe [nvm](#) repo on. GitHub
- Node Package Manager (npm) — Siehe [npm-Website](#).

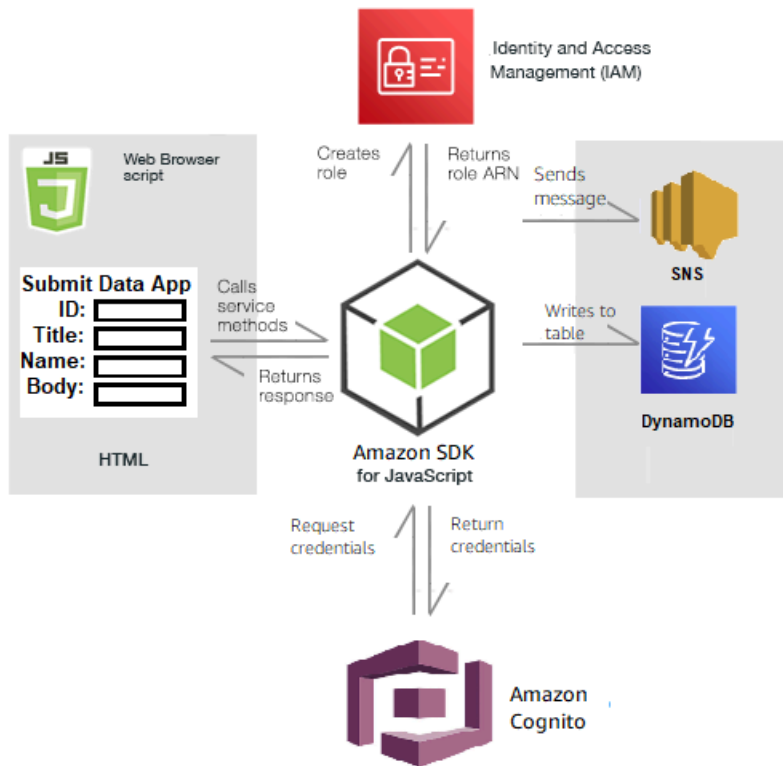
Erstellen Sie eine App zum Senden von Daten an DynamoDB

Dieses serviceübergreifende Tutorial zu Node.js zeigt, wie Sie eine App erstellen, mit der Benutzer Daten an eine Amazon DynamoDB-Tabelle senden können. Diese App verwendet die folgenden Dienste:

- AWS Identity and Access Management(IAM) und Amazon Cognito für Autorisierung und Berechtigungen.
- Amazon DynamoDB (DynamoDB), um die Tabellen zu erstellen und zu aktualisieren.
- Amazon Simple Notification Service (Amazon SNS), um den App-Administrator zu benachrichtigen, wenn ein Benutzer die Tabelle aktualisiert.

Das Szenario

In diesem Tutorial bietet eine HTML-Seite eine browserbasierte Anwendung zum Senden von Daten an eine Amazon DynamoDB-Tabelle. Die App verwendet Amazon SNS, um den App-Administrator zu benachrichtigen, wenn ein Benutzer die Tabelle aktualisiert.



Um die App zu erstellen:

1. [Voraussetzungen](#)
2. [Bereitstellen von Ressourcen](#)
3. [Erstelle den HTML-Code](#)
4. [Erstellen Sie das Browser-Skript](#)
5. [Nächste Schritte](#)

Voraussetzungen

Führen Sie die folgenden erforderlichen Aufgaben aus:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Erstellen Sie die Ressourcen AWS

Diese App benötigt die folgenden Ressourcen:

- AWS Identity and Access Management(IAM) Nicht authentifizierte Amazon Cognito Cognito-Benutzerrolle mit den folgenden Berechtigungen:
 - sns:Publish
 - Dynamodb: PutItem
- Eine DynamoDB-Tabelle.

Sie können diese Ressourcen manuell in der AWS Konsole erstellen, wir empfehlen jedoch, diese Ressourcen AWS CloudFormation wie in diesem Tutorial beschrieben bereitzustellen.

Erstellen Sie die AWS Ressourcen mit AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation Benutzerhandbuch](#).

Um den AWS CloudFormation Stack mit dem zu erstellenAWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLIBenutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

Note

Die AWS CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und ersetzen Sie *STACK_NAME* durch einen eindeutigen Namen für den Stack und *REGION in Ihrer Region*. AWS

Important

Der Stack-Name muss innerhalb einer AWS Region und eines Kontos eindeutig sein. AWS Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM --region REGION
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLIBefehlsreferenz](#) und im [AWS CloudFormationBenutzerhandbuch](#).

Um die erstellten Ressourcen anzuzeigen, öffnen AWS CloudFormation Sie die AWS Verwaltungskonsole, wählen Sie den Stack und dann die Registerkarte Ressourcen aus.

4. Verwenden Sie bei der Erstellung des Stacks die, AWS SDK for JavaScript um die DynamoDB-Tabelle aufzufüllen, wie unter beschrieben. [Auffüllen der Tabelle](#)

Auffüllen der Tabelle

Um die Tabelle zu füllen, erstellen Sie zunächst ein Verzeichnis mit dem Namen `libs` und darin eine Datei mit dem Namen `DynamoClient.js` , und fügen Sie den folgenden Inhalt ein. Ersetzen Sie *REGION* durch Ihre AWS Region und ersetzen Sie *IDENTITY_POOL_ID durch eine Amazon Cognito Identity Pool-ID*. Dadurch wird das DynamoDB-Clientobjekt erstellt.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";  
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";  
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```



```
const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { dynamoClient };
```

Dieser Code ist [hier auf verfügbar. GitHub](#)

Erstellen Sie als Nächstes einen `dynamoAppHelperFiles` Ordner in Ihrem Projektordner, erstellen Sie eine Datei `update-table.js` darin und kopieren Sie den Inhalt [hier GitHub](#) hinein.

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { dynamoClient } from "../libs/dynamoClient.js";

// Set the parameters
export const params = {
  TableName: "Items",
  Item: {
    id: { N: "1" },
    title: { S: "aTitle" },
    name: { S: "aName" },
    body: { S: "aBody" },
  },
};

export const run = async () => {
  try {
    const data = await dynamoClient.send(new PutItemCommand(params));
    console.log("success");
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}
```

```
};  
run();
```

Führen Sie den folgenden Befehl von der Befehlszeile aus.

```
node update-table.js
```

Dieser Code ist [hier auf](#) verfügbar GitHub.

Erstellen Sie eine Front-End-Seite für die App

Hier erstellen Sie die Frontend-HTML-Browserseite für die App.

Erstellen Sie ein DynamoDBApp Verzeichnis, erstellen Sie eine Datei mit dem Namen `index.html` und kopieren Sie von [hier an den GitHub](#) Code. Das `script` Element fügt die `main.js` Datei hinzu, die alle JavaScript für das Beispiel erforderlichen Informationen enthält. Sie werden die `main.js` Datei später in diesem Tutorial erstellen. Mit dem verbleibenden Code `index.html` wird die Browserseite erstellt, auf der die von Benutzern eingegebenen Daten erfasst werden.

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Erstellen Sie das Browser-Skript

Erstellen Sie zunächst die für das Beispiel erforderlichen Service-Client-Objekte. Erstellen Sie ein `libs` Verzeichnisses `Client.js`, erstellen Sie es und fügen Sie den folgenden Code ein. Ersetzen Sie *jeweils* `REGION` und `IDENTITY_POOL_ID`.

Note

Verwenden Sie die ID des Amazon Cognito Cognito-Identitätspools, in [Erstellen Sie die Ressourcen AWS](#) dem Sie ihn erstellt haben.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";  
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";  
import { SNSClient } from "@aws-sdk/client-sns";  
  
const REGION = "REGION";  
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.
```

```
// Create an Amazon Comprehend service client object.
const snsClient = new SNSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { snsClient };
```

Dieser Code ist [hier auf GitHub verfügbar](#) .

Um das Browserskript für dieses Beispiel zu erstellen, erstellen Sie in einem Ordner mit dem Namen ein Modul Node.js mit dem Dateinamen `add_data.js` und fügen Sie den folgenden Code ein. Die `submitData` Funktion sendet Daten an eine DynamoDB-Tabelle und sendet mithilfe von Amazon SNS einen SMS-Text an den App-Administrator.

Deklariieren Sie in der `submitData` Funktion Variablen für die Zieltelefonnummer, die auf der App-Oberfläche eingegebenen Werte und für den Namen des Amazon S3 S3-Buckets. Erstellen Sie als Nächstes ein Parameter-Objekt zum Hinzufügen eines Elements zur Tabelle. Wenn keiner der Werte leer ist, `submitData` fügt das Element der Tabelle hinzu und sendet die Nachricht. Denken Sie daran, die Funktion für den Browser verfügbar zu machen, mit `window.submitData = submitData`.

```
// Import required AWS SDK clients and commands for Node.js
import { PutItemCommand } from "@aws-sdk/client-dynamodb";
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
import { dynamoClient } from "../libs/dynamoClient.js";

export const submitData = async () => {
  //Set the parameters
  // Capture the values entered in each field in the browser (by id).
  const id = document.getElementById("id").value;
  const title = document.getElementById("title").value;
  const name = document.getElementById("name").value;
  const body = document.getElementById("body").value;
  //Set the table name.
  const tableName = "Items";

  //Set the parameters for the table
```

```
const params = {
  TableName: tableName,
  // Define the attributes and values of the item to be added. Adding ' + "" '
  // converts a value to
  // a string.
  Item: {
    id: { N: id + "" },
    title: { S: title + "" },
    name: { S: name + "" },
    body: { S: body + "" },
  },
};
// Check that all the fields are completed.
if (id !== "" && title !== "" && name !== "" && body !== "") {
  try {
    //Upload the item to the table
    await dynamoClient.send(new PutItemCommand(params));
    alert("Data added to table.");
    try {
      // Create the message parameters object.
      const messageParams = {
        Message: "A new item with ID value was added to the DynamoDB",
        PhoneNumber: "PHONE_NUMBER", //PHONE_NUMBER, in the E.164 phone number
        // structure.
        // For example, a standard local formatted number, such as (415) 555-2671,
        // is +14155552671 in E.164
        // format, where '1' is the country code.
      };
      // Send the SNS message
      const data = await snsClient.send(new PublishCommand(messageParams));
      console.log(
        "Success, message published. MessageID is " + data.MessageId,
      );
    } catch (err) {
      // Display error message if error is not sent
      console.error(err, err.stack);
    }
  } catch (err) {
    // Display error message if item is not added to table
    console.error(
      "An error occurred. Check the console for further information",
      err,
    );
  }
}
```

```
// Display alert if all field are not completed.
} else {
  alert("Enter data in each field.");
}
};
// Expose the function to the browser
window.submitData = submitData;
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Führen Sie abschließend an der Befehlszeile den folgenden Befehl aus, um das JavaScript für dieses Beispiel in einer Datei mit dem Namen `main.js` zu bündeln:

```
webpack add_data.js --mode development --target web --devtool false -o main.js
```

Note

Hinweise zur Installation von Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#).

Um die App auszuführen, öffnen Sie `index.html` in Ihrem Browser.

Löschen Sie die Ressourcen

Wie zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen zu beenden, die Sie während der Bearbeitung dieses Tutorials erstellen, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird. Sie können dies tun, indem Sie den AWS CloudFormation Stack, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie das [AWS CloudFormation in der AWS Managementkonsole](#).
2. Öffnen Sie die Seite Stacks und wählen Sie den Stack aus.
3. Wählen Sie Löschen aus.

Weitere AWS dienstübergreifende Beispiele finden Sie unter [AWS SDK for JavaScript serviceübergreifende](#) Beispiele.

Erstellen Sie eine Transkriptions-App mit authentifizierten Benutzern

In diesem Kurs lernen Sie Folgendes:

- Implementieren Sie die Authentifizierung mithilfe eines Amazon Cognito Cognito-Identitätspools, um Benutzer zu akzeptieren, die mit einem Amazon Cognito Cognito-Benutzerpool verbunden sind.
- Verwenden Sie Amazon Transcribe, um Sprachaufnahmen zu transkribieren und im Browser anzuzeigen.

Das Szenario

Die App ermöglicht es Benutzern, sich mit einer eindeutigen E-Mail-Adresse und einem eindeutigen Benutzernamen anzumelden. Nach Bestätigung ihrer E-Mail können sie Sprachnachrichten aufzeichnen, die automatisch transkribiert und in der App angezeigt werden.

Funktionsweise

Die App verwendet zwei Amazon S3 S3-Buckets, einen zum Hosten des Anwendungscodes und einen zum Speichern von Transkriptionen. Die App verwendet einen Amazon-Cognito-Benutzerpool zur Authentifizierung Ihrer Benutzer. Authentifizierte Benutzer verfügen über IAM-Berechtigungen für den Zugriff auf die erforderlichen Dienste. AWS

Wenn ein Benutzer zum ersten Mal eine Sprachnachricht aufzeichnet, erstellt Amazon S3 einen eindeutigen Ordner mit dem Namen des Benutzers im Amazon S3 S3-Bucket zum Speichern von Transkriptionen. Amazon Transcribe transkribiert die Sprachnachricht in Text und speichert sie in JSON im Ordner des Benutzers. Wenn der Benutzer die App aktualisiert, werden seine Transkriptionen angezeigt und können heruntergeladen oder gelöscht werden.

Für dieses Tutorial benötigen Sie ungefähr 30 Minuten.

Schritte

Um die App zu erstellen:

1. [Voraussetzungen](#)
2. [Erstellen Sie die AWS Ressourcen](#)
3. [Erstellen Sie den HTML-Code](#)
4. [Bereiten Sie das Browser-Skript vor](#)
5. [Führen Sie die App aus](#)
6. [Löschen Sie die Ressourcen](#)

Voraussetzungen

- Richten Sie die Projektumgebung ein, um diese JavaScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJS-Syntax](#)

Erstellen Sie die Ressourcen AWS

In diesem Abschnitt wird beschrieben, wie Sie AWS Ressourcen für diese App mithilfe von bereitstellenAWS Cloud Development Kit (AWS CDK).

Note

Das AWS CDK ist ein Softwareentwicklungsframework, mit dem Sie Cloud-Anwendungsressourcen definieren können. Weitere Informationen finden Sie im [AWS Cloud Development Kit \(AWS CDK\)-Entwicklerhandbuch](#).

Um Ressourcen für die App zu erstellen, verwenden Sie die Vorlage [hier weiter](#), GitHub um einen AWS CDK Stack entweder mit der [AWSWeb Services Management Console](#) oder dem zu erstellen [AWS CLI](#). Anweisungen zum Ändern des Stacks oder zum Löschen des Stacks und der zugehörigen Ressourcen, wenn Sie das Tutorial abgeschlossen haben, finden Sie [hier GitHub](#).

Note

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

Der resultierende Stapel stellt automatisch die folgenden Ressourcen bereit.

- Ein Amazon Cognito Cognito-Identitätspool mit einer authentifizierten Benutzerrolle.
- Eine IAM-Richtlinie mit Berechtigungen für Amazon S3 und Amazon Transcribe ist an die Rolle des authentifizierten Benutzers angehängt.
- Ein Amazon Cognito Cognito-Benutzerpool, der es Benutzern ermöglicht, sich bei der App zu registrieren und anzumelden.
- Ein Amazon S3 S3-Bucket zum Hosten der Anwendungsdateien.
- Ein Amazon S3 S3-Bucket zum Speichern der Transkriptionen.

Important

Dieser Amazon S3 S3-Bucket ermöglicht den öffentlichen Zugriff READ (LIST), sodass jeder die Objekte innerhalb des Buckets auflisten und die Informationen möglicherweise missbrauchen kann. Wenn Sie diesen Amazon S3-Bucket nicht unmittelbar nach Abschluss des Tutorials löschen, empfehlen wir Ihnen dringend, die [Best Practices für Sicherheit in Amazon S3](#) im Amazon Simple Storage Service-Benutzerhandbuch zu befolgen.

Erstellen Sie den HTML-Code

Erstellen Sie eine `index.html` Datei, kopieren Sie den folgenden Inhalt und fügen Sie ihn ein. Die Seite enthält ein Bedienfeld mit Schaltflächen zum Aufzeichnen von Sprachnachrichten und eine Tabelle, in der die zuvor transkribierten Nachrichten des aktuellen Benutzers angezeigt werden. Das Skript-Tag am Ende des `body` Elements ruft die `main.js`, die das gesamte Browser-Skript für die App enthalten. Sie erstellen die Datei `main.js` mithilfe von Webpack, wie im folgenden Abschnitt dieses Tutorials beschrieben.

```
<!DOCTYPE html>
<html>
<head>
```



```
<meta charset="UTF-8">
<title>title</title>
<link rel="stylesheet" type="text/css" href="recorder.css">
<style>
  table, td {
    border: 1px solid black;
  }
</style>
</head>
<body>
<h2>Record</h2>
<p>
  <button id="record" onclick="startRecord()"></button>
  <button id="stopRecord" disabled onclick="stopRecord()">Stop</button>
<p id="demo" style="visibility: hidden;"></p>
</p>
<p>
  <audio id="recordedAudio"></audio>
</p>

<h2>My transcriptions</h2>
<table id="myTable1" style = "width:678px;">
</table>
<table id="myTable" style = "width:678px;">
  <tr>
    <td style = "font-weight:bold">Time created</td>
    <td style = "font-weight:bold">Transcription</td>
    <td style = "font-weight:bold">Download</td>
    <td style = "font-weight:bold">Delete</td>
  </tr>
</table>

<script type="text/javascript" src="./main.js"></script>
</body>

</html>
```

Dieses Codebeispiel ist [hier unter verfügbar GitHub](#).

Bereiten Sie das Browser-Skript vor

Es gibt drei Dateien, `index.html`, `recorder.js`, und `helper.js`, die Sie `main.js` mithilfe von Webpack zu einer einzigen bündeln müssen. In diesem Abschnitt werden nur die Funktionen

detailliert beschrieben `index.js`, für die das SDK verwendet wird JavaScript, das [hier verfügbar](#) [GitHub](#) ist.

Note

`recorder.js` und `helper.js` sind erforderlich, werden aber, da sie keinen Node.js Code enthalten, in den Inline-Kommentaren [hier](#) bzw. [hier](#) erklärt GitHub.

Definieren Sie zunächst die Parameter. `COGNITO_ID` ist der Endpunkt für den Amazon Cognito Cognito-Benutzerpool, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben. Er ist formatiert `cognito-idp.AWS_REGION.amazonaws.com/USER_POOL_ID`. Die Benutzerpool-ID ist `ID_TOKEN` im AWS Anmeldeinformationstoken, das durch die `getToken` Funktion in der Datei 'helper.js' aus der App-URL entfernt wird. Dieses Token wird an die `loginData` Variable übergeben, die den Amazon Transcribe- und Amazon S3 S3-Client-Objekten Logins zur Verfügung stellt. Ersetzen Sie „`REGION`“ durch die AWS Region und „`BUCKET`“ durch das *Ersetzen Sie „`IDENTITY_POOL_ID`“* durch das `IdentityPoolId` von der Beispielseite des Amazon Cognito Cognito-Identitätspools, das Sie für dieses Beispiel erstellt haben. Dies wird auch an jedes Client-Objekt übergeben.

```
// Import the required AWS SDK clients and commands for Node.js
import "./helper.js";
import "./recorder.js";
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import {
  CognitoIdentityProviderClient,
  GetUserCommand,
} from "@aws-sdk/client-cognito-identity-provider";
import { S3RequestPresigner } from "@aws-sdk/s3-request-presigner";
import { createRequest } from "@aws-sdk/util-create-request";
import { formatUrl } from "@aws-sdk/util-format-url";
import {
  TranscribeClient,
  StartTranscriptionJobCommand,
} from "@aws-sdk/client-transcribe";
import {
  S3Client,
  PutObjectCommand,
  GetObjectCommand,
```

```
ListObjectsCommand,
DeleteObjectCommand,
} from "@aws-sdk/client-s3";
import fetch from "node-fetch";

// Set the parameters.
// 'COGNITO_ID' has the format 'cognito-idp.eu-west-1.amazonaws.com/COGNITO_ID'.
let COGNITO_ID = "COGNITO_ID";
// Get the Amazon Cognito ID token for the user. 'getToken()' is in 'helper.js'.
let idToken = getToken();
let loginData = {
  [COGNITO_ID]: idToken,
};

const params = {
  Bucket: "BUCKET", // The Amazon Simple Storage Solution (S3) bucket to store the
  transcriptions.
  Region: "REGION", // The AWS Region
  identityPoolID: "IDENTITY_POOL_ID", // Amazon Cognito Identity Pool ID.
};

// Create an Amazon Transcribe service client object.
const client = new TranscribeClient({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});

// Create an Amazon S3 client object.
const s3Client = new S3Client({
  region: params.Region,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: params.Region }),
    identityPoolId: params.identityPoolID,
    logins: loginData,
  }),
});
```

Wenn die HTML-Seite geladen wird, `updateUserInterface` erstellt der einen Ordner mit dem Namen des Benutzers im Amazon S3 S3-Bucket, falls er sich zum ersten Mal bei der App anmeldet. Wenn nicht, aktualisiert es die Benutzeroberfläche mit allen Abschriften der vorherigen Sitzungen des Benutzers.

```
window.onload = async () => {
  // Set the parameters.
  const userParams = {
    // Get the access token. 'GetAccessToken()' is in 'helper.js'.
    AccessToken: getAccessToken(),
  };
  // Create a CognitoIdentityProviderClient client object.
  const client = new CognitoIdentityProviderClient({ region: params.Region });
  try {
    const data = await client.send(new GetUserCommand(userParams));
    const username = data.Username;
    // Export username for use in 'recorder.js'.
    exports.username = username;
    try {
      // If this is user's first sign-in, create a folder with user's name in Amazon S3
      bucket.
      // Otherwise, no effect.
      const Key = `${username}/`;
      try {
        const data = await s3Client.send(
          new PutObjectCommand({ Key: Key, Bucket: params.Bucket })
        );
        console.log("Folder created for user ", data.Username);
      } catch (err) {
        console.log("Error", err);
      }
    }
    try {
      // Get a list of the objects in the Amazon S3 bucket.
      const data = await s3Client.send(
        new ListObjectsCommand({ Bucket: params.Bucket, Prefix: username })
      );
      // Create a variable for the list of objects in the Amazon S3 bucket.
      const output = data.Contents;
      // Loop through the objects, populating a row on the user interface for each
      object.
      for (var i = 0; i < output.length; i++) {
        var obj = output[i];
        const objectParams = {
```

```
    Bucket: params.Bucket,
    Key: obj.Key,
  };
  // Get the name of the object from the Amazon S3 bucket.
  const data = await s3Client.send(new GetObjectCommand(objectParams));
  // Extract the body contents, a readable stream, from the returned data.
  const result = data.Body;
  // Create a variable for the string version of the readable stream.
  let stringResult = "";
  // Use 'yieldUnit8Chunks' to convert the readable streams into JSON.
  for await (let chunk of yieldUnit8Chunks(result)) {
    stringResult += String.fromCharCode.apply(null, chunk);
  }
  // The setTimeout function waits while readable stream is converted into
JSON.

  setTimeout(function () {
    // Parse JSON into human readable transcript, which will be displayed on
user interface (UI).
    const outputJSON =
      JSON.parse(stringResult).results.transcripts[0].transcript;
    // Create name for transcript, which will be displayed.
    const outputJSONTime = JSON.parse(stringResult)
      .jobName.split("/")[0]
      .replace("-job", "");
    i++;
    //
    // Display the details for the transcription on the UI.
    // 'displayTranscriptionDetails()' is in 'helper.js'.
    displayTranscriptionDetails(
      i,
      outputJSONTime,
      objectParams.Key,
      outputJSON
    );
  }, 1000);
}
} catch (err) {
  console.log("Error", err);
}
} catch (err) {
  console.log("Error creating presigned URL", err);
}
} catch (err) {
  console.log("Error", err);
}
```

```
    }  
};  
  
// Convert readable streams.  
async function* yieldUint8Chunks(data) {  
  const reader = data.getReader();  
  try {  
    while (true) {  
      const { done, value } = await reader.read();  
      if (done) return;  
      yield value;  
    }  
  } finally {  
    reader.releaseLock();  
  }  
}
```

Wenn der Benutzer eine Sprachnachricht für Transkriptionen aufzeichnet, upload lädt er die Aufzeichnungen in den Amazon S3 S3-Bucket hoch. Diese Funktion wird aus der Datei aufgerufen. `recorder.js`

```
// Upload recordings to Amazon S3 bucket  
window.upload = async function (blob, userName) {  
  // Set the parameters for the recording recording.  
  const Key = `${userName}/test-object-${Math.ceil(Math.random() * 10 ** 10)}`;  
  let signedUrl;  
  
  // Create a presigned URL to upload the transcription to the Amazon S3 bucket when it  
  // is ready.  
  try {  
    // Create an Amazon S3RequestPresigner object.  
    const signer = new S3RequestPresigner({ ...s3Client.config });  
    // Create the request.  
    const request = await createRequest(  
      s3Client,  
      new PutObjectCommand({ Key, Bucket: params.Bucket })  
    );  
    // Define the duration until expiration of the presigned URL.  
    const expiration = new Date(Date.now() + 60 * 60 * 1000);  
    // Create and format the presigned URL.  
    signedUrl = formatUrl(await signer.presign(request, expiration));  
  }  
}
```

```
    console.log(`\nPutting "${Key}"`);
  } catch (err) {
    console.log("Error creating presigned URL", err);
  }
  try {
    // Upload the object to the Amazon S3 bucket using a presigned URL.
    response = await fetch(signedUrl, {
      method: "PUT",
      headers: {
        "content-type": "application/octet-stream",
      },
      body: blob,
    });
    // Create the transcription job name. In this case, it's the current date and time.
    const today = new Date();
    const date =
      today.getFullYear() +
      "-" +
      (today.getMonth() + 1) +
      "-" +
      today.getDate();
    const time =
      today.getHours() + "-" + today.getMinutes() + "-" + today.getSeconds();
    const jobName = date + "-time-" + time;

    // Call the "createTranscriptionJob()" function.
    createTranscriptionJob(
      "s3://" + params.Bucket + "/" + Key,
      jobName,
      params.Bucket,
      Key
    );
  } catch (err) {
    console.log("Error uploading object", err);
  }
};

// Create the AWS Transcribe transcription job.
const createTranscriptionJob = async (recording, jobName, bucket, key) => {
  // Set the parameters for transcriptions job
  const params = {
    TranscriptionJobName: jobName + "-job",
    LanguageCode: "en-US", // For example, 'en-US',
    OutputBucketName: bucket,
```

```
    OutputKey: key,
    Media: {
      MediaFileUri: recording, // For example, "https://transcribe-demo.s3-
REGION.amazonaws.com/hello_world.wav"
    },
  };
  try {
    // Start the transcription job.
    const data = await client.send(new StartTranscriptionJobCommand(params));
    console.log("Success - transcription submitted", data);
  } catch (err) {
    console.log("Error", err);
  }
};
```

`deleteTranscription` löscht eine Transkription von der Benutzeroberfläche und `deleteRow` löscht eine bestehende Transkription aus dem Amazon S3 S3-Bucket. Beide werden durch die Schaltfläche Löschen auf der Benutzeroberfläche ausgelöst.

```
// Delete a transcription from the Amazon S3 bucket.
window.deleteJSON = async (jsonFileName) => {
  try {
    await s3Client.send(
      new DeleteObjectCommand({
        Bucket: params.Bucket,
        Key: jsonFileName,
      })
    );
    console.log("Success - JSON deleted");
  } catch (err) {
    console.log("Error", err);
  }
};

// Delete a row from the user interface.
window.deleteRow = function (rowid) {
  const row = document.getElementById(rowid);
  row.parentNode.removeChild(row);
};
```


Führen Sie abschließend an der Eingabeaufforderung den folgenden Befehl aus, um die Datei JavaScript für dieses Beispiel in einer Datei mit dem Namen `main.js` zu bündeln:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Note

Hinweise zur Installation von Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#).

Führen Sie die App aus

Sie können die App dann an der folgenden Stelle anzeigen.

```
DOMAIN/login?  
client_id=APP_CLIENT_ID&response_type=token&scope=aws.cognito.signin.user.admin+email  
+openid+phone+profile&redirect_uri=REDIRECT_URL
```

Amazon Cognito erleichtert die Ausführung der App, indem es einen Link in der AWS Web Services Management Console bereitstellt. Navigieren Sie einfach zur App-Client-Einstellung Ihres Amazon Cognito Cognito-Benutzerpools und wählen Sie Launch Hosted UI aus. Die URL für die App hat das folgende Format.

Important

Die gehostete Benutzeroberfläche verwendet standardmäßig den Antworttyp „Code“. Dieses Tutorial ist jedoch für den Antworttyp „Token“ konzipiert, sodass Sie ihn ändern müssen.

Löschen Sie die AWS Ressourcen

Wenn Sie das Tutorial abgeschlossen haben, sollten Sie die Ressourcen löschen, damit Ihnen keine unnötigen Kosten entstehen. Da Sie Inhalt zu beiden Amazon S3 S3-Buckets hinzugefügt haben, müssen Sie sie manuell löschen. Anschließend können Sie die verbleibenden Ressourcen entweder mit der [AWS Web Services Management Console](#) oder mit dem [AWS CLI](#) löschen. Anweisungen zum Ändern des Stacks oder zum Löschen des Stacks und der zugehörigen Ressourcen, wenn Sie das Tutorial abgeschlossen haben, finden Sie [hier GitHub](#).

Lambda mit API Gateway aufrufen

Sie können eine Lambda-Funktion mithilfe von Amazon API Gateway aufrufen. Dabei handelt es sich um einen AWS Service zum Erstellen, Veröffentlichen, Verwalten, Überwachen und Sichern von REST, HTTP und WebSocket APIs in großem Umfang. API-Entwickler können APIs erstellen, die auf AWS oder andere Web-Services sowie auf Daten zugreifen können, die in der AWS Cloud gespeichert sind. Als API Gateway Gateway-Entwickler können Sie APIs für die Verwendung in Ihren eigenen Client-Anwendungen erstellen. Weitere Informationen finden Sie unter [Was ist Amazon API Gateway](#).

AWS Lambda ist ein Rechenservice, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Sie können Lambda-Funktionen in verschiedenen Programmiersprachen erstellen. Weitere Informationen zu AWS Lambda finden Sie unter [Was ist AWS Lambda?](#)

In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. Dieses Beispiel ruft verschiedene AWS-Dienste auf, um einen bestimmten Anwendungsfall zu erfüllen. Nehmen wir beispielsweise an, dass eine Organisation ihren Mitarbeitern eine mobile Textnachricht sendet, in der sie zum einjährigen Jubiläum gratuliert, wie in dieser Abbildung gezeigt.



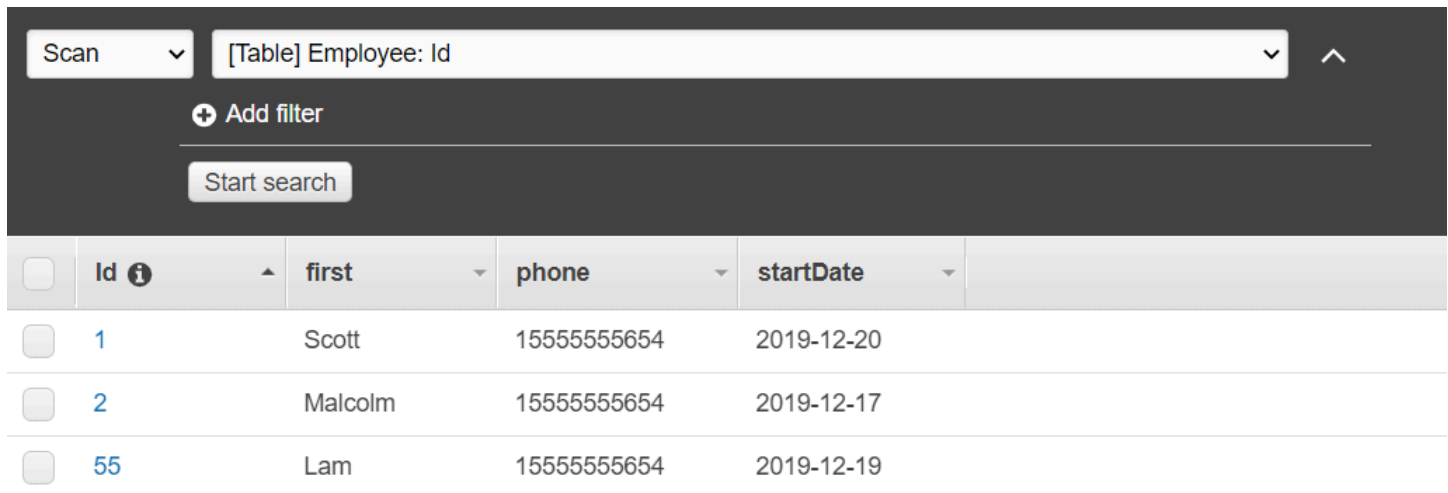
Die Fertigstellung des Beispiels sollte etwa 20 Minuten in Anspruch nehmen.

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe von JavaScript Logik eine Lösung erstellen, die diesen Anwendungsfall erfüllt. Sie lernen beispielsweise, eine Datenbank zu lesen, um festzustellen, welche Mitarbeiter das einjährige Jubiläum erreicht haben, wie die Daten verarbeitet und eine Textnachricht mithilfe einer Lambda-Funktion versendet werden. Anschließend erfahren Sie, wie Sie API Gateway verwenden, um diese AWS Lambda Funktion mithilfe eines Rest-Endpunkts aufzurufen. Sie können die Lambda-Funktion beispielsweise mit dem folgenden curl-Befehl aufrufen:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/employee"
```

AWSIn diesem Tutorial wird eine Amazon DynamoDB-Tabelle mit dem Namen Employee verwendet, die diese Felder enthält.

- id — der Primärschlüssel für die Tabelle.
- FirstName — Vorname des Mitarbeiters.
- Telefon — Telefonnummer des Mitarbeiters.
- StartDate — Startdatum des Mitarbeiters.



<input type="checkbox"/>	Id ⓘ	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Bearbeitungskosten: Die in diesem Dokument enthaltenen AWS Dienste sind im AWS kostenlosen Kontingent enthalten. Achten Sie jedoch darauf, alle Ressourcen zu beenden, nachdem Sie dieses Beispiel abgeschlossen haben, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird.

Um die App zu erstellen:

1. [Vollständige Voraussetzungen](#)
2. [Erstellen Sie die AWS Ressourcen](#)
3. [Bereiten Sie das Browser-Skript vor](#)
4. [Lambda-Funktion erstellen und hochladen](#)
5. [Stellen Sie die Lambda-Funktion bereit](#)
6. [Führen Sie die App aus](#)

7. [Löschen Sie die Ressourcen](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Erstellen Sie die Ressourcen AWS

Für dieses Tutorial sind die folgenden Ressourcen erforderlich:

- Eine Amazon DynamoDB-Tabelle, benannt Employee mit einem Schlüssel Id und den Feldern, die in der vorherigen Abbildung gezeigt wurden. Stellen Sie sicher, dass Sie die richtigen Daten eingeben, einschließlich eines gültigen Mobiltelefons, mit dem Sie diesen Anwendungsfall testen möchten. Weitere Informationen finden Sie unter [Tabelle erstellen](#).
- Eine IAM-Rolle mit angehängten Berechtigungen zur Ausführung von Lambda-Funktionen.
- Ein Amazon S3 S3-Bucket zum Hosten der Lambda-Funktion.

Sie können diese Ressourcen manuell erstellen, wir empfehlen jedoch, diese Ressourcen AWS CloudFormation wie in diesem Tutorial beschrieben bereitzustellen.


Erstellen Sie die AWS Ressourcen mit AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation Benutzerhandbuch](#).

Um den AWS CloudFormation Stack mit dem zu erstellen AWS CLI:


1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLIBenutzerhandbuch](#).

- Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

 Note

Die AWS CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

- Führen Sie den folgenden Befehl von der Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

 Important

Der Stack-Name muss innerhalb einer AWS Region und eines Kontos eindeutig sein. AWS Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLIBefehlsreferenz](#) und im [AWS CloudFormationBenutzerhandbuch](#).

- Füllen Sie als Nächstes die Tabelle wie folgt [Füllen der Tabelle](#) aus.

Füllen der Tabelle

Um die Tabelle zu füllen, erstellen Sie zunächst ein Verzeichnis mit dem Namen `libs` und darin eine Datei mit dem Namen `DynamoClient.js` , und fügen Sie den folgenden Inhalt ein.

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Dieser Code ist [hier verfügbar GitHub](#).

Erstellen Sie als Nächstes eine Datei mit dem Namen `populate-table.js` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein. Ersetzen Sie für eines der Elemente den Wert für die `phone` Eigenschaft durch eine gültige Handynummer im E.164-Format und den Wert für `startDate` durch das heutige Datum.

Führen Sie den folgenden Befehl von der Befehlszeile aus.

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "../libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
```

```
        id: { N: "55" },
        firstName: { S: "Harriette" },
        phone: { N: "155555555555652" },
        startDate: { S: "2019-12-19" },
    },
},
],
},
};

export const run = async () => {
    try {
        const data = await dbclient.send(new BatchWriteItemCommand(params));
        console.log("Success", data);
    } catch (err) {
        console.log("Error", err);
    }
};
run();
```

Dieser Code ist [hier auf](#) verfügbar GitHub.

Erstellen der AWS Lambda-Funktion

Konfigurieren des SDKs

Erstellen Sie im `libs` Verzeichnis Dateien mit dem Namen `snsClient.js` und `lambdaClient.js` und fügen Sie den folgenden Inhalt jeweils in diese Dateien ein.

```
const { SNSClient } = require ( "@aws-sdk/client-sns" );
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon SNS service client object.
const snsClient = new SNSClient({ region: REGION });
module.exports = { snsClient };
```

Ersetzen Sie **REGION** durch die AWS Region. Dieser Code ist [hier verfügbar GitHub](#).

```
const { LambdaClient } = require ( "@aws-sdk/client-lambda" );
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

Ersetzen Sie **REGION** durch die AWS Region. Dieser Code ist [hier verfügbar GitHub](#).

Importieren Sie zunächst die erforderlichen AWS SDK for JavaScript (v3) Module und Befehle. Dann berechne das heutige Datum und weise es einem Parameter zu. Drittens erstellen Sie die Parameter für den ScanCommand. Ersetzen Sie **TABLE_NAME** durch den Namen der Tabelle, die Sie im [Erstellen Sie die Ressourcen AWS](#) Abschnitt dieses Beispiels erstellt haben.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```
"use strict";
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const {snsClient} = require ( "./libs/snsClient" );
const {dynamoClient} = require ( "./libs/dynamoClient" );

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```


Scannen der DynamoDB-Tabelle

Erstellen Sie zunächst eine Async/Await-Funktion, die aufgerufen wird `sendText`, um eine Textnachricht mithilfe von Amazon SNS zu veröffentlichen. `PublishCommand` Fügen Sie dann ein `try` Blockmuster hinzu, das die DynamoDB-Tabelle nach Mitarbeitern durchsucht, deren heutiges Arbeitsjubiläum ansteht. Anschließend wird die `sendText` Funktion aufgerufen, diesen Mitarbeitern eine Textnachricht zu senden. Wenn ein Fehler auftritt, wird der `catch` Block aufgerufen.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    data.Items.forEach(function (element) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Bündelung der Lambda-Funktion

In diesem Thema wird beschrieben, wie Sie die `myLambdafunction.ts` und die erforderlichen AWS SDK for JavaScript Module für dieses Beispiel in einer gebündelten Datei namens `bündeln.index.js`

1. Falls Sie es noch nicht getan haben, folgen Sie den Anweisungen [Erforderliche Aufgaben](#) für dieses Beispiel, um Webpack zu installieren.

Note

Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#)

2. Führen Sie in der Befehlszeile den folgenden Befehl aus, um das JavaScript für dieses Beispiel in einer Datei namens `<index.js>` zu bündeln:

```
webpack mylambdafunction.ts --mode development --target node --devtool false --output-library-target umd -o index.js
```

Important

Beachten Sie, dass die Ausgabe benannt ist `index.js`. Dies liegt daran, dass Lambda-Funktionen einen `index.js` Handler haben müssen, um zu funktionieren.

3. Komprimieren Sie die gebündelte Ausgabedatei, `index.js`, in eine ZIP-Datei mit dem Namen `mylambdafunction.zip`
4. Laden Sie `mylambdafunction.zip` es in den Amazon S3 S3-Bucket hoch, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben.

Stellen Sie die Lambda-Funktion bereit

Erstellen Sie im Stammverzeichnis Ihres Projekts eine `lambda-function-setup.ts` Datei und fügen Sie den folgenden Inhalt ein.

Ersetzen Sie `BUCKET_NAME` durch den Namen des Amazon S3 S3-Buckets, in den Sie die ZIP-Version Ihrer Lambda-Funktion hochgeladen haben. Ersetzen Sie `ZIP_FILE_NAME` durch den Namen der ZIP-Version Ihrer Lambda-Funktion. Ersetzen Sie `ROLE` durch die Amazon-Ressourcennummer (ARN) der IAM-Rolle, die Sie im [Erstellen Sie die Ressourcen AWS](#) Thema

dieses Tutorials erstellt haben. Ersetzen Sie *LAMBDA_FUNCTION_NAME* durch einen Namen für die Lambda-Funktion.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient } = require ( "../libs/lambdaClient.js" );

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
    Services (Amazon SNS) to " +
    "send employees an email on each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Geben Sie in der Befehlszeile Folgendes ein, um die Lambda-Funktion bereitzustellen.

```
node lambda-function-setup.ts
```

Dieses Codebeispiel ist [hier verfügbar GitHub](#).

API Gateway so konfigurieren, dass die Lambda-Funktion aufgerufen wird

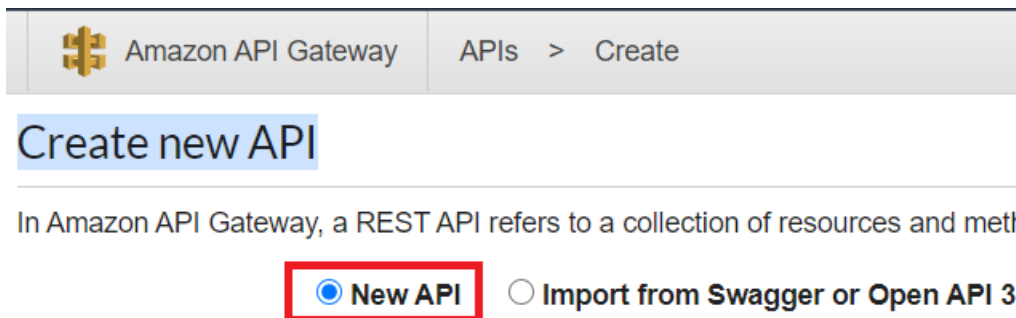
Um die App zu erstellen:

1. [Erstellen Sie die Rest-API](#)
2. [Testen Sie die API Gateway Gateway-Methode](#)
3. [Stellen Sie die API Gateway Gateway-Methode bereit](#)

Erstellen Sie die Rest-API

Sie können die API Gateway Gateway-Konsole verwenden, um einen Rest-Endpunkt für die Lambda-Funktion zu erstellen. Sobald Sie fertig sind, können Sie die Lambda-Funktion mit einem RESTful-Aufruf aufrufen.

1. Melden Sie sich bei der [Amazon API Gateway Gateway-Konsole](#) an.
2. Wählen Sie unter Rest API die Option Build aus.
3. Wählen Sie Neue API aus.



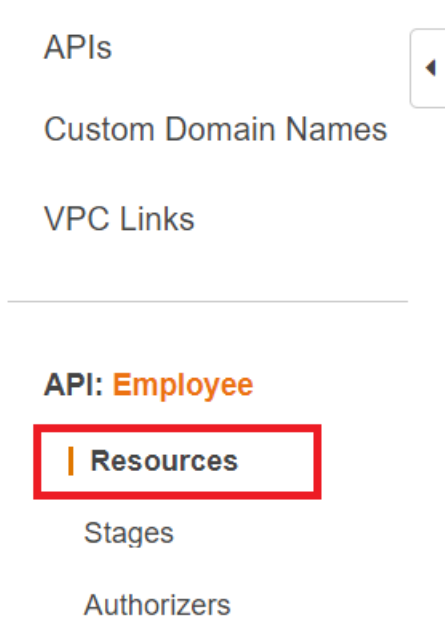
4. Geben Sie Employee als API-Namen an und geben Sie eine Beschreibung ein.

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="Employee"/>
Description	<input type="text" value="This invokes a Lambda function"/>
Endpoint Type	<input type="text" value="Regional"/> ▼ ⓘ

- Wählen Sie Create API (API erstellen) aus.
- Wählen Sie im Bereich Mitarbeiter die Option Ressourcen aus.



- Geben Sie im Namensfeld Mitarbeiter an.
- Wählen Sie Create Resources (Ressourcen erstellen) aus.
- Wählen Sie im Drop-down-Menü Aktionen die Option Ressourcen erstellen aus.

Use this page to create a new child resource for your resource. 🟡

Configure as [proxy resource](#)

 ⓘ

Resource Name*

Resource Path*

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS

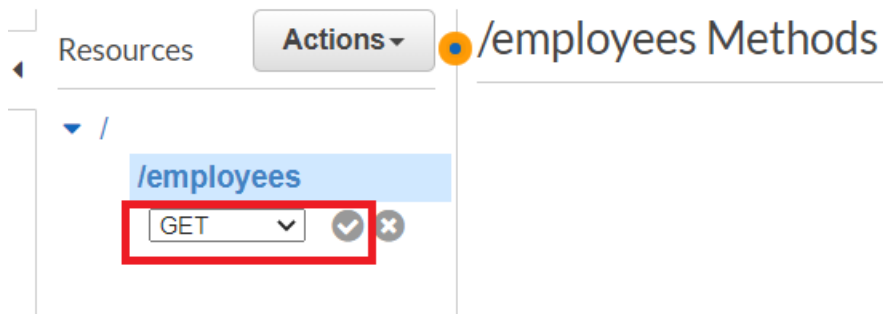
 ⓘ

* Required

Cancel

Create Resource

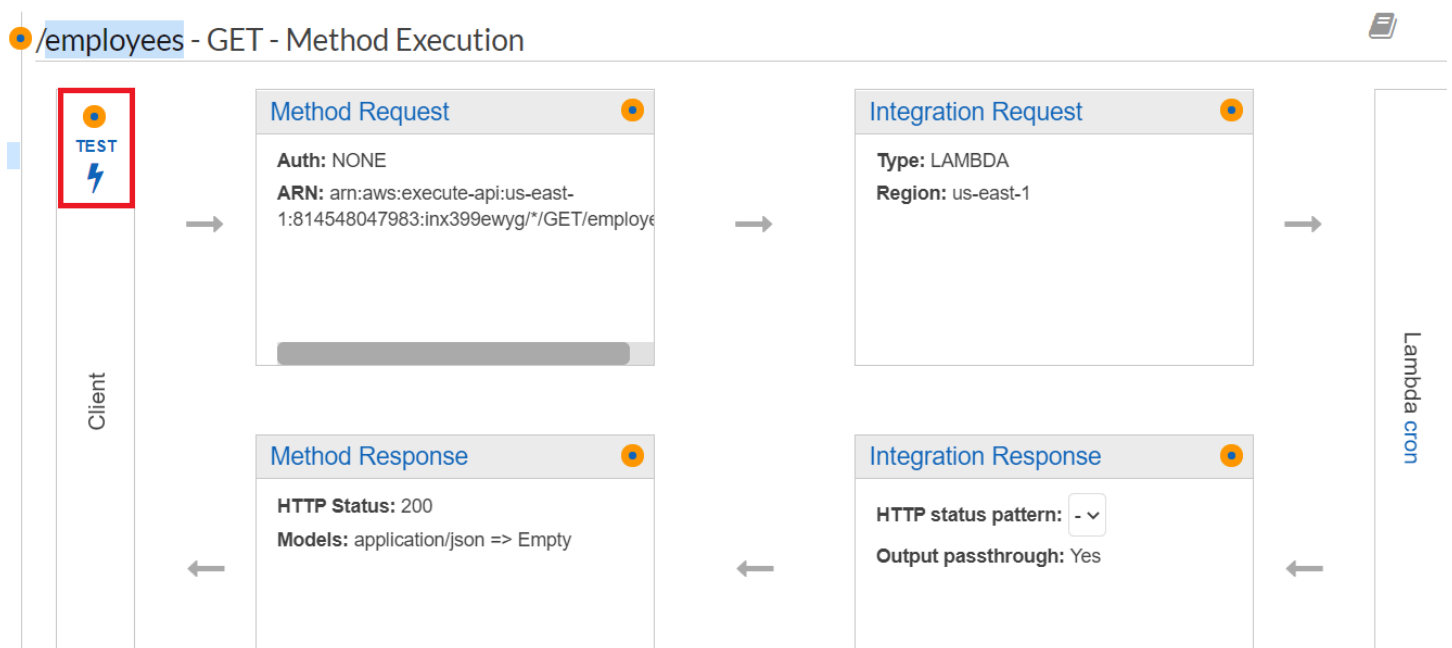
- Wählen Sie `/employees` aus, wählen Sie in den Aktionen die Option Methode erstellen aus und wählen Sie anschließend im Drop-down-Menü unter `/employees` die Option GET aus. Wählen Sie das Häkchen aus.



- Wählen Sie Lambda-Funktion und geben Sie mylambdafunction als Namen der Lambda-Funktion ein. Wählen Sie Speichern aus.

Testen Sie die API Gateway Gateway-Methode

An dieser Stelle des Tutorials können Sie die API Gateway Gateway-Methode testen, die die Lambda-Funktion mylambdafunction aufruft. Um die Methode zu testen, wählen Sie Test aus, wie in der folgenden Abbildung gezeigt.

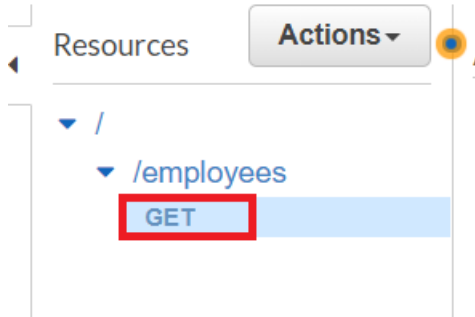


Sobald die Lambda-Funktion aufgerufen wurde, können Sie die Protokolldatei einsehen, um eine Erfolgsmeldung zu sehen.

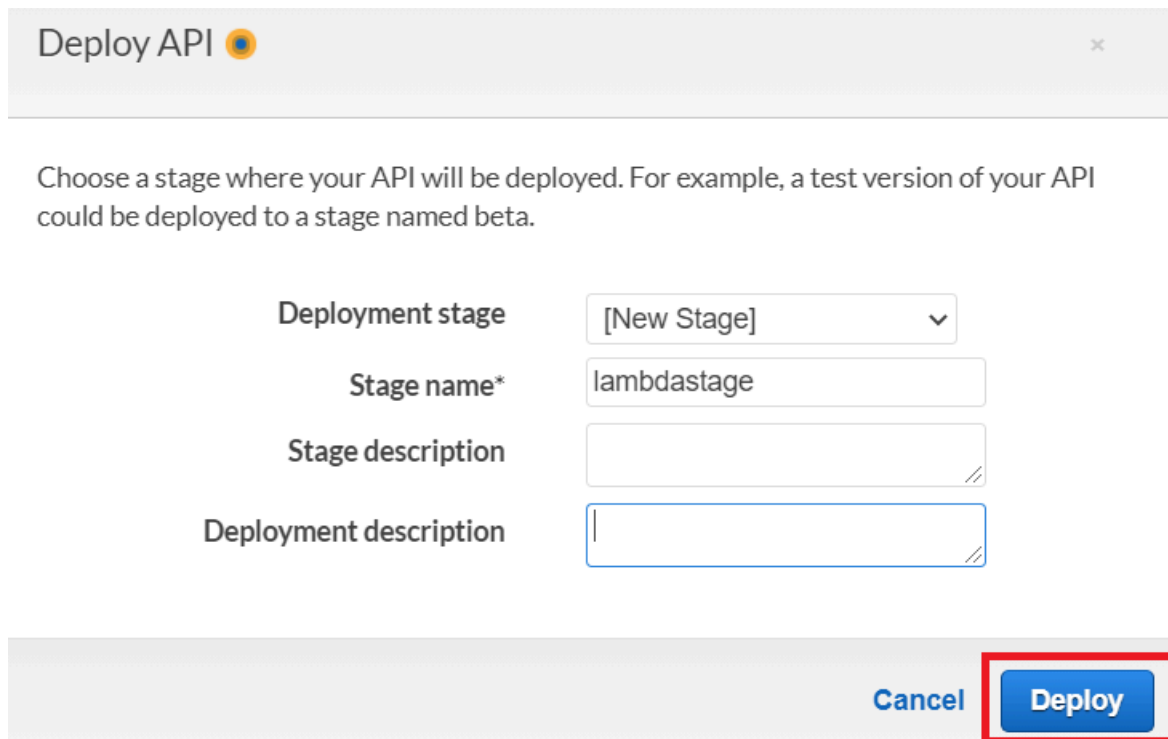
Stellen Sie die API Gateway Gateway-Methode bereit

Nach erfolgreichem Test können Sie die Methode über die [Amazon API Gateway Gateway-Konsole](#) bereitstellen.

1. Wählen Sie Get.



2. Wählen Sie im Drop-down-Menü „Aktionen“ die Option API bereitstellen aus.

A screenshot of the 'Deploy API' dialog box. The title bar reads 'Deploy API' with a close button (x) on the right. Below the title bar is a text instruction: 'Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.' The form contains four fields: 'Deployment stage' is a dropdown menu with '[New Stage]' selected; 'Stage name*' is a text input field containing 'lambdastage'; 'Stage description' is a text area; and 'Deployment description' is a text area. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Deploy'. The 'Deploy' button is highlighted with a red rectangular box.

3. Füllen Sie das Formular Deploy API aus und wählen Sie Deploy aus.

Deploy API ✕

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	<input type="text" value="[New Stage]"/>
Stage name*	<input type="text" value="lambdastage"/>
Stage description	<input type="text"/>
Deployment description	<input type="text"/>

Cancel
Deploy

4. Wählen Sie Save Changes.
5. Wählen Sie erneut Abrufen und stellen Sie fest, dass sich die URL ändert. Dies ist die Aufruf-URL, mit der Sie die Lambda-Funktion aufrufen können.

Stages
Create

- lambdastage
- /
- /employees
- GET

lambdastage - GET - /employees

Invoke URL: [https://\[redacted\].execute-api.us-east-1.amazonaws.com/lambdastage/employees](https://[redacted].execute-api.us-east-1.amazonaws.com/lambdastage/employees)

Use this page to override the `lambdastage` stage settings for the GET to /employees method.

Settings Inherit from stage
 Override for this method

Löschen Sie die Ressourcen

Herzlichen Glückwunsch! Sie haben eine Lambda-Funktion über Amazon API Gateway mit dem aufgerufen. AWS SDK for JavaScript Wie bereits zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen zu beenden, die Sie während der Bearbeitung dieses Tutorials erstellen, um sicherzustellen, dass Ihnen nichts berechnet wird. Sie können dies tun, indem Sie den AWS CloudFormation Stack, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie das [AWS CloudFormation in der AWS Managementkonsole](#).

2. Öffnen Sie die Seite Stacks und wählen Sie den Stack aus.
3. Wählen Sie Löschen.

AWSServerlose Workflows erstellen mit AWS SDK for JavaScript

Sie können einen AWS serverlosen Workflow erstellen, indem Sie Step Functions, das AWS SDK for Java und AWS Step Functions verwenden. Jeder Workflow-Schritt wird mit einer AWS Lambda-Funktion implementiert. Lambda ist ein Datenverarbeitungsservice, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Step Functions ist ein Serverless-Orchestrierungsservice, mit dem Sie Lambda-Funktionen und andere kombinieren AWS-Services kombinieren können, um geschäftskritische Anwendungen zu erstellen.

Note

Sie können Lambda-Funktionen in verschiedenen Programmiersprachen erstellen. In diesem Tutorial wurden Lambda-Funktionen mithilfe der Lambda-Java-API implementiert. Weitere Informationen zu Lambda finden Sie unter [Was ist Lambda](#).

In diesem Tutorial erstellen Sie einen Workflow, der Supporttickets für eine Organisation erstellt. Jeder Workflow-Schritt führt einen Vorgang mit dem Ticket aus. Dieses Tutorial zeigt Ihnen, wie Sie Workflow-Daten verarbeiten können. JavaScript Sie lernen beispielsweise, wie Sie Daten lesen, die an den Workflow übergeben wurden, wie Sie Daten zwischen Schritten weitergeben und wie Sie AWS Dienste aus dem Workflow aufrufen.

Kosten für den Abschluss: Die in diesem Dokument enthaltenen AWS Dienste sind im [AWSkostenlosen Kontingent](#) enthalten.

Hinweis: Stellen Sie sicher, dass Sie alle Ressourcen, die Sie während der Bearbeitung dieses Tutorials erstellt haben, beenden, um sicherzustellen, dass Ihnen nichts mehr berechnet wird.

Themen

- [Erforderliche Aufgaben](#)
- [Erstellen Sie die Ressourcen AWS](#)
- [Den Workflow erstellen](#)
- [Erstellen Sie die Lambda-Funktionen](#)
- [Fügen Sie die Lambda-Funktionen zu Workflows hinzu](#)

- [Führen Sie Ihren Workflow mithilfe der Step Functions-Konsole aus](#)
- [Löschen Sie die Ressourcen AWS](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Erstellen Sie die Ressourcen AWS

Für dieses Tutorial sind die folgenden Ressourcen erforderlich.

- Eine Amazon DynamoDB-Tabelle namens Case mit einem Schlüssel namens Id.
- Eine IAM-Rolle mit dem Namen, die zum Aufrufen von Lambda-Funktionen `lambda-support` verwendet wird. Diese Rolle verfügt über Richtlinien, die es ihr ermöglichen, die Amazon DynamoDB- und Amazon Simple Email Service-Dienste von einer Lambda-Funktion aus aufzurufen.
- Eine IAM-Rolle mit dem Namen, die zum Aufrufen des Workflows `workflow-support` verwendet wird.
- Ein Amazon S3 S3-Bucket zum Hosten der Lambda-Funktionen.


Sie können diese Ressourcen manuell erstellen, wir empfehlen jedoch, diese Ressourcen mithilfe von AWS Cloud Development Kit (AWS CDK) (AWS CDK) bereitzustellen, wie in diesem Tutorial beschrieben.

Erstellen Sie die AWS Ressourcen mit dem AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation Benutzerhandbuch](#).


Um den AWS CloudFormation Stack zu erstellen:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLIBenutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

 Note

Die AWS CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

 Important

Der Stack-Name muss innerhalb einer AWS Region und eines Kontos eindeutig sein. AWS Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLIBefehlsreferenz](#) und im [AWS CloudFormationBenutzerhandbuch](#).

Erstellen Sie die AWS Ressourcen mithilfe der Amazon Web Services Management Console;

Folgen Sie den Anweisungen im [AWS CloudFormationBenutzerhandbuch](#), um Ressourcen für die App in der Konsole zu erstellen. Verwenden Sie die bereitgestellte Vorlagesetup.yaml, erstellen Sie eine Datei mit dem Namen und kopieren Sie den Inhalt [hier weiter GitHub](#).

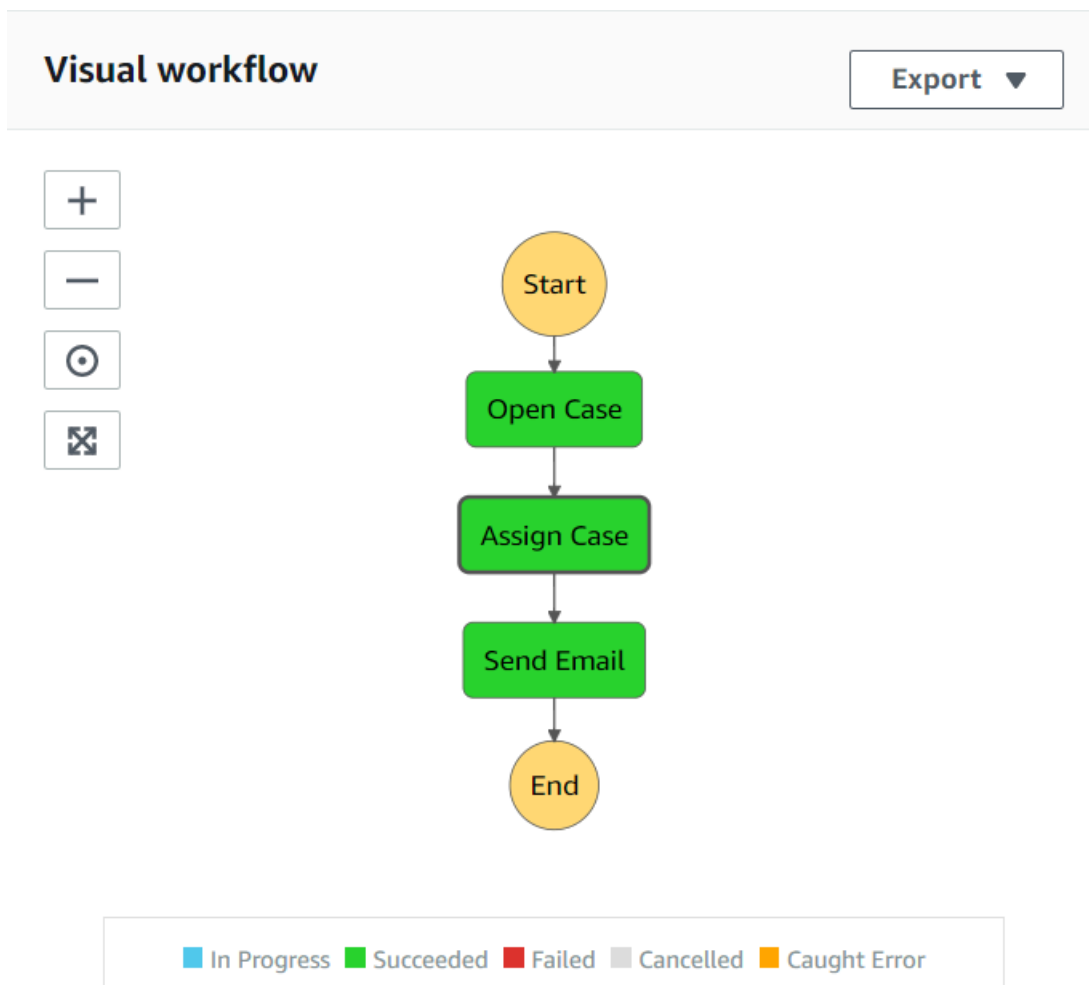
⚠ Important

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

Rufen Sie eine Liste der Ressourcen in der Konsole auf, indem Sie den Stack im AWS CloudFormation Dashboard öffnen und die Registerkarte Ressourcen auswählen. Sie benötigen diese für das Tutorial.

Den Workflow erstellen

Die folgende Abbildung zeigt den Workflow, den Sie mit diesem Tutorial erstellen werden.



Folgendes passiert bei jedem Schritt des Workflows:

+ Start — Initiiert den Workflow.

- + Fall öffnen — Verarbeitet den ID-Wert eines Supporttickets, indem er an den Workflow weitergegeben wird.
- + Fall zuweisen — Weist den Support-Fall einem Mitarbeiter zu und speichert die Daten in einer DynamoDB-Tabelle.
- + E-Mail senden — Sendet dem Mitarbeiter mithilfe des Amazon Simple Email Service (Amazon SES) eine E-Mail-Nachricht, um ihn darüber zu informieren, dass ein neues Ticket vorliegt.
- + Ende — Stoppt den Workflow.

Erstellen Sie mithilfe von Step-Funktionen einen serverlosen Workflow

Sie können einen Workflow erstellen, der Supporttickets verarbeitet. Um einen Workflow mithilfe von Step Functions zu definieren, erstellen Sie ein Dokument in Amazon States Language (JSON-basiert), um Ihren State Machine zu definieren. Ein Sprachdokument von Amazon States beschreibt jeden Schritt. Nachdem Sie das Dokument definiert haben, bietet Step Functions eine visuelle Darstellung des Workflows. Die folgende Abbildung zeigt das Sprachdokument von Amazon States und die visuelle Darstellung des Workflows.

Workflows können Daten zwischen Schritten weitergeben. Beispielsweise verarbeitet der Schritt „Kundenvorgang öffnen“ einen Fall-ID-Wert (der an den Workflow übergeben wird) und übergibt diesen Wert an den Schritt „Fall zuweisen“. Später in diesem Tutorial erstellen Sie Anwendungslogik in der Lambda-Funktion, um die Datenwerte zu lesen und zu verarbeiten.

So erstellen Sie ein Workflow

1. Öffnen Sie die [Amazon Web Services Console](#).
2. Wählen Sie Create State Machine aus.
3. Wählen Sie Author with code snippets (Autor mit Codeausschnitten) aus. Wählen Sie im Bereich Typ die Option Standard aus.

Define state machine

Author with code snippets
Author your workflow using Amazon States Language. You can generate code snippets to easily build out your workflow steps.

Run a sample project
Deploy and run a fully functioning sample project in minutes using CloudFormation.

Start with a template
Get started quickly with common patterns for Amazon States Language.

Type

Standard
Durable, checkpointed workflows for machine learning, order fulfillment, IT/DevOps automation, ETL jobs, and other long-duration workflows.

Express New
Event-driven workflows for streaming data processing, microservices orchestration, IoT data ingestion, mobile backends, and other short duration, high-event-rate workflows.

► [Help me decide](#)

4. Geben Sie das Sprachdokument für die Amazon-Staaten an, indem Sie den folgenden Code eingeben.

```
{
  "Comment": "A simple AWS Step Functions state machine that automates a call center support session.",
  "StartAt": "Open Case",
  "States": {
    "Open Case": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
      "Next": "Assign Case"
    },
    "Assign Case": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
      "Next": "Send Email"
    },
    "Send Email": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
      "End": true
    }
  }
}
```

Note

Machen Sie sich keine Sorgen über die Fehler im Zusammenhang mit den Lambda-Ressourcenwerten. Sie werden diese Werte später in diesem Tutorial aktualisieren.

5. Wählen Sie Weiter.
6. Geben Sie im Namensfeld ein SupportStateMachine.
7. Wählen Sie im Abschnitt Permission die Option Choose an existing role aus.
8. Wählen Sie workflow-support (die IAM-Rolle, die Sie erstellt haben).

Permissions

Execution role
The IAM role that defines which resources your state machine has permission to access during execution. To create a custom role, go to the [IAM console](#).

Create new role
Let Step Functions create a new role for you based on your state machine's definition and configuration details.

Choose an existing role

Enter a role ARN

Existing roles

workflow-support

9. Wählen Sie Create State Machine (Zustandsautomaten erstellen). Es wird eine Meldung angezeigt, die besagt, dass die Zustandsmaschine erfolgreich erstellt wurde.

✔ State machine successfully created

Step Functions > State machines > SupportStateMachine

SupportStateMachine

Edit Start execution Delete Actions

Details	
ARN	arn:aws:states:us-west-2:81454...:stateMachine:SupportStateMachine
Type	Standard
IAM role ARN	arn:aws:iam::81454...:role/service-role/StepFunctions-SupportStateMachine-role
Creation date	Jun 1, 2020 02:53:16.855 PM

Executions | Logging | Definition | Tags

Erstellen Sie die Lambda-Funktionen

Verwenden Sie die Lambda-Laufzeit-API, um die Lambda-Funktionen zu erstellen. In diesem Beispiel gibt es drei Workflow-Schritte, die jeweils jeder Lambda-Funktion entsprechen.

Erstellen Sie diese Lambda-Funktionen, wie in den folgenden Abschnitten beschrieben:

- [GetID Lambda-Funktion](#)- Wird als erster Schritt im Workflow verwendet, der den Ticket-ID-Wert verarbeitet.
- [AddItem Lambda-Klasse](#)- Wird als zweiter Schritt im Workflow verwendet, der das Ticket einem Mitarbeiter zuweist und die Daten in einer DynamoDB-Datenbank speichert.
- [sendemail Lambda-Klasse](#)- Wird als dritter Schritt im Workflow verwendet, bei dem Amazon SES eine E-Mail-Nachricht an den Mitarbeiter gesendet wird, um ihn über das Ticket zu informieren.

GetID Lambda-Funktion

Erstellen Sie eine Lambda-Funktion, die den Ticket-ID-Wert zurückgibt, der an den zweiten Schritt im Workflow übergeben wird.

```
exports.handler = async (event) => {
  // Create a support case using the input as the case ID, then return a confirmation
  message
  try{
    const myCaseID = event.inputCaseID;
    var myMessage = "Case " + myCaseID + ": opened...";
    var result = { Case: myCaseID, Message: myMessage };
  }
  catch(err){
    console.log('Error', err);
  }
};
```

Geben Sie Folgendes in die Befehlszeile ein, um die Datei mithilfe von Webpack in einer Datei mit dem Namen zu bündeln. `index.js`

```
webpack getid.js --mode development --target node --devtool false --output-library-
target umd -o index.js
```


Komprimieren Sie dann `index.js` in einen ZIP Dateinamen `id.js.zip`. Laden Sie die ZIP Datei in den Amazon S3 S3-Bucket hoch, den Sie im Thema dieses Beispiels erstellt haben.

Dieses Codebeispiel ist [hier unter](#) verfügbar GitHub.

AddItem Lambda-Klasse

Erstellen Sie eine Lambda-Funktion, die einen Mitarbeiter auswählt, dem das Ticket zugewiesen wird, und die Ticketdaten dann in einer DynamoDB-Tabelle mit dem Namen `Case` speichert.

```
"use strict";
// Load the required clients and commands.
const { PutItemCommand } = require ( "@aws-sdk/client-dynamodb" );
const { dynamoClient } = require ( "../libs/dynamoClient" );

exports.handler = async (event) => {
  try {
    // Helper function to send message using Amazon SNS.
    const val = event;
    //PersistCase adds an item to a DynamoDB table
    const tmp = Math.random() <= 0.5 ? 1 : 2;
    console.log(tmp);
    if (tmp == 1) {
      const params = {
        TableName: "Case",
        Item: {
          id: { N: val.Case },
          empEmail: { S: "brmur@amazon.com" },
          name: { S: "Tom Blue" },
        },
      };
      console.log("adding item for tom");
      try {
        const data = await dynamoClient.send(new PutItemCommand(params));
        console.log(data);
      } catch (err) {
        console.error(err);
      }
      var result = { Email: params.Item.empEmail };
      return result;
    } else {
      const params = {
        TableName: "Case",
```

```
    Item: {
      id: { N: val.Case },
      empEmail: { S: "RECEIVER_EMAIL_ADDRESS" }, // Valid Amazon Simple
Notification Services (Amazon SNS) email address.
      name: { S: "Sarah White" },
    },
  };
console.log("adding item for sarah");
try {
  const data = await dynamoClient.send(new PutItemCommand(params));
  console.log(data);
} catch (err) {
  console.error(err);
}
return params.Item.empEmail;
var result = { Email: params.Item.empEmail };
}
} catch (err) {
  console.log("Error", err);
}
};
```

Geben Sie in der Befehlszeile Folgendes ein, um die Datei mithilfe von Webpack in einer Datei mit dem Namen zu bündeln. `index.js`

```
webpack additem.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

Komprimieren Sie dann `index.js` in einen ZIP Dateinamen `additem.js.zip`. Laden Sie die ZIP Datei in den Amazon S3 S3-Bucket hoch, den Sie im Thema dieses Beispiels erstellt haben.

Dieses Codebeispiel ist [hier unter](#) verfügbar GitHub.

sendemail Lambda-Klasse

Erstellen Sie eine Lambda-Funktion, die eine E-Mail sendet, um sie über das neue Ticket zu informieren. Die E-Mail-Adresse, die im zweiten Schritt übergeben wurde, wird verwendet.

```
// Load the required clients and commands.
const { SendEmailCommand } = require ( "@aws-sdk/client-ses" );
const { sesClient } = require ( "../libs/sesClient" );
```

```
exports.handler = async (event) => {
  // Enter a sender email address. This address must be verified.
  const senderEmail = "SENDER_EMAIL"
  const sender = "Sender Name <" + senderEmail + ">";

  // AWS Step Functions passes the employee's email to the event.
  // This address must be verified.
  const recipient = event.S;

  // The subject line for the email.
  const subject = "New case";

  // The email body for recipients with non-HTML email clients.
  const body_text =
    "Hello,\r\n" + "Please check the database for new ticket assigned to you.";

  // The HTML body of the email.
  const body_html = `<head></head><body><h1>Hello!</h1><p>Please check the
database for new ticket assigned to you.</p></body></html>`;

  // The character encoding for the email.
  const charset = "UTF-8";
  var params = {
    Source: sender,
    Destination: {
      ToAddresses: [recipient],
    },
    Message: {
      Subject: {
        Data: subject,
        Charset: charset,
      },
      Body: {
        Text: {
          Data: body_text,
          Charset: charset,
        },
        Html: {
          Data: body_html,
          Charset: charset,
        },
      },
    },
  },
};
```

```
try {
  const data = await sesClient.send(new SendEmailCommand(params));
  console.log(data);
} catch (err) {
  console.error(err);
}
};
```

Geben Sie in der Befehlszeile Folgendes ein, um die Datei mithilfe von Webpack in einer Datei mit dem Namen `index.js` zu bündeln.

```
webpack sendemail.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

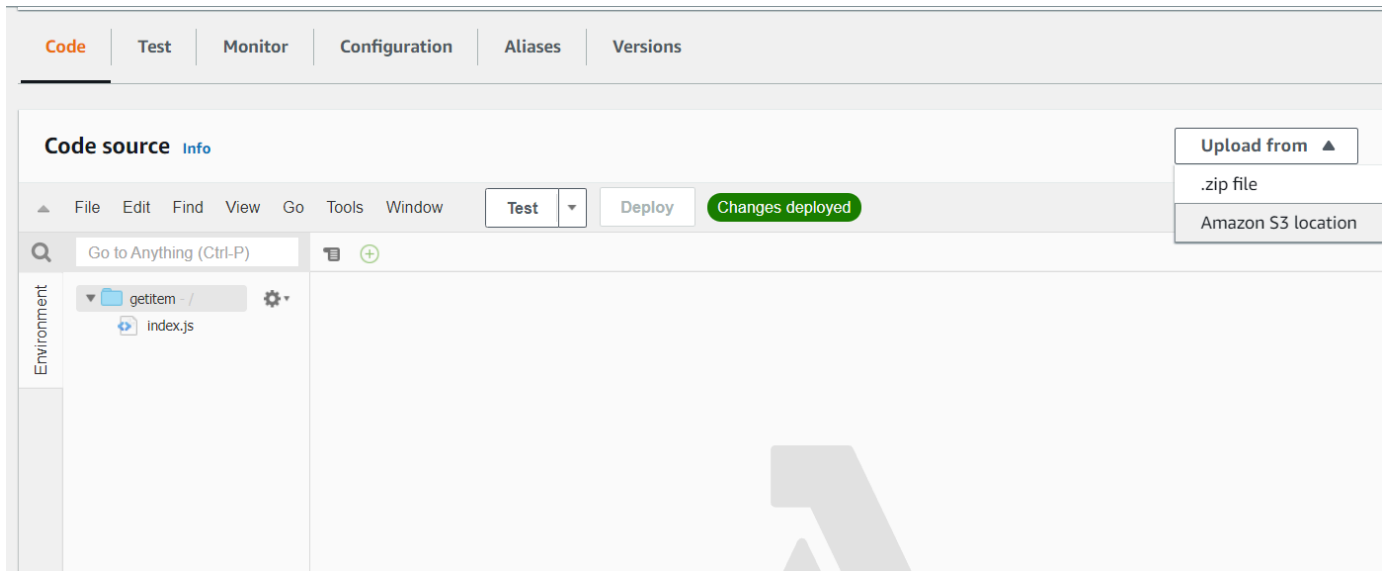
Komprimieren Sie dann `index.js` in einen ZIP Dateinamens `sendemail.js.zip`. Laden Sie die ZIP Datei in den Amazon S3 S3-Bucket hoch, den Sie im Thema dieses Beispiels erstellt haben.

Dieses Codebeispiel ist [hier unter](#) verfügbar GitHub.

Stellen Sie die Lambda-Funktionen bereit

So stellen Sie die `getid` Lambda-Funktion bereit:

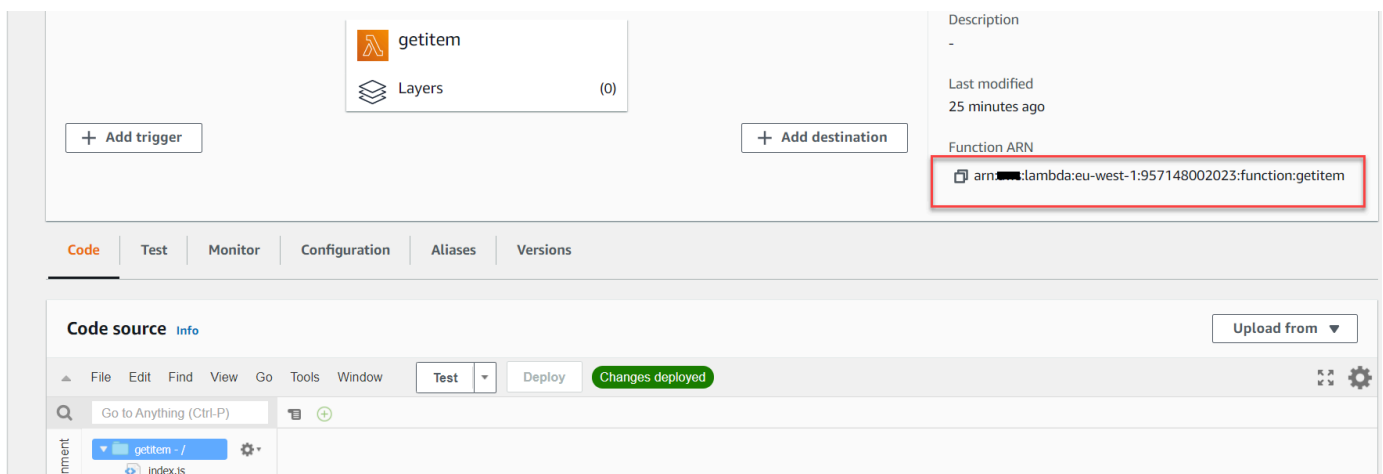
1. Öffnen Sie die Lambda-Konsole in der [Amazon Web Services Console](#).
2. Wählen Sie Funktion erstellen.
3. Wählen Sie Von Grund auf neu schreiben aus.
4. Geben Sie im Abschnitt Grundinformationen `getid` als Namen ein.
5. Wählen Sie in der Runtime `Node.js 14x` aus.
6. Wählen Sie Eine bestehende Rolle verwenden und anschließend Lambda-Support (die IAM-Rolle, die Sie in der erstellt haben) aus.
7. Wählen Sie Funktion erstellen.
8. wählen Sie Upload von — Amazon S3 S3-Standort.
9. Wählen Sie Upload, wählen Sie Upload von — Amazon S3 S3-Standort und geben Sie die Amazon S3 S3-Link-URL ein.



10. Wählen Sie Speichern aus.
11. Wiederholen Sie diesen Vorgang für die neuen Lambda-Funktionen additem.js.zip und sendemail.js.zip. Wenn Sie fertig sind, stehen Ihnen drei Lambda-Funktionen zur Verfügung, auf die Sie im Sprachdokument von Amazon States verweisen können.

Fügen Sie die Lambda-Funktionen zu Workflows hinzu

1. Öffnen Sie die Lambda-Konsole. Beachten Sie, dass Sie den Wert Lambda Amazon Resource Name (ARN) in der oberen rechten Ecke sehen können.



2. Kopieren Sie den Wert und fügen Sie ihn dann in Schritt 1 des Sprachdokuments Amazon States ein, das sich in der Step Functions-Konsole befindet.

3. Aktualisieren Sie die Ressource für die Schritte „Fall zuweisen“ und „E-Mail senden“. So binden Sie Lambda-Funktionen, die mit dem AWS SDK for Java erstellt wurden, in einen Workflow ein, der mithilfe von Step Functions erstellt wurde.

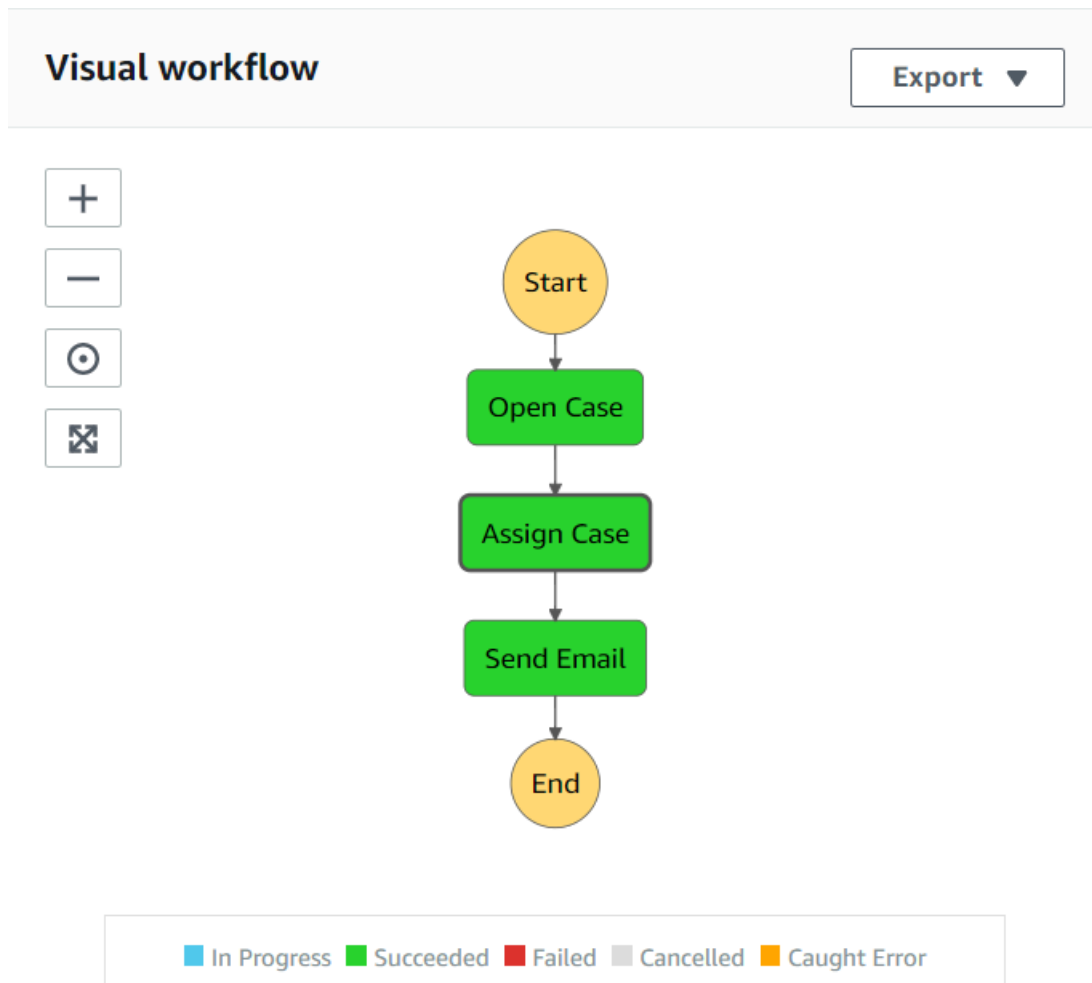
Führen Sie Ihren Workflow mithilfe der Step Functions-Konsole aus

Sie können den Workflow in der Step Functions-Konsole aufrufen. Eine Ausführung empfängt JSON-Eingaben. In diesem Beispiel können Sie die folgenden JSON-Daten an den Workflow übergeben.

```
{  
  "inputCaseID": "001"  
}
```

Um Ihren Workflow auszuführen:

1. Wählen Sie in der Step Functions Functions-Konsole die Option Ausführung starten aus.
2. Übergeben Sie im Abschnitt Eingabe die JSON-Daten. Sehen Sie sich den Workflow an. Wenn jeder Schritt abgeschlossen ist, wird er grün.



3. Wenn der Schritt rot wird, ist ein Fehler aufgetreten. Sie können auf den Schritt klicken und die Protokolle anzeigen, auf die von der rechten Seite aus zugegriffen werden kann.

Code
Step details

Name	Type
Assign Case	Task

Status

✔ Succeeded

Resource

arn:aws:lambda:us-west-2:8145-... :function:function3 [↗](#) CloudWatch

[logs ↗](#)

- ▶ Input
- ▶ Output
- ▶ Exception

Wenn der Workflow abgeschlossen ist, können Sie die Daten in der DynamoDB-Tabelle anzeigen.

Scan: [\[Table\] Case: id](#) ^

Scan ▾ [Table] Case: id ▾

+ Add filter

Start search

	id ⓘ	email	name	registrationDate
<input type="checkbox"/>	001	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	091	swhite@noServer.com	Sarah White	1586217600
<input type="checkbox"/>	111	tblue@noServer.com	Tom Blue	1586217600
<input type="checkbox"/>	888	swhite@noServer.com	Sarah White	1586217600

Löschen Sie die Ressourcen AWS

Herzlichen Glückwunsch, Sie haben mit dem AWS SDK for Java einen AWS serverlosen Workflow erstellt. Wie zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen zu beenden, die Sie während der Bearbeitung dieses Tutorials erstellen, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird. Sie können dies tun, indem Sie den AWS CloudFormation Stack, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie das [AWS CloudFormation in der AWS Managementkonsole](#).
2. Öffnen Sie die Seite Stacks und wählen Sie den Stack aus.
3. Wählen Sie Löschen.

Erstellen geplanter Ereignisse zum Ausführen von -AWS LambdaFunktionen

Sie können ein geplantes Ereignis erstellen, das eine -AWS LambdaFunktion aufruft, indem Sie ein Amazon- CloudWatch Ereignis verwenden. Sie können ein CloudWatch Ereignis so konfigurieren, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann eine Lambda-Funktion aufgerufen wird. Sie können beispielsweise ein CloudWatch Ereignis so planen, dass jeden Wochentag eine Lambda-Funktion aufgerufen wird.

AWS Lambda ist ein Datenverarbeitungsservice, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Sie können Lambda-Funktionen in verschiedenen Programmiersprachen erstellen. Weitere Informationen zu AWS Lambda finden Sie unter [Was ist AWS Lambda?](#)

In diesem Tutorial erstellen Sie eine Lambda-Funktion mithilfe der Lambda JavaScript-Laufzeit-API. Dieses Beispiel ruft verschiedene AWS-Dienste auf, um einen bestimmten Anwendungsfall zu erfüllen. Angenommen, eine Organisation sendet eine mobile Textnachricht an ihre Mitarbeiter, die ihnen zum einjährigen Jubiläum gratuliert, wie in dieser Abbildung gezeigt.



Für dieses Tutorial benötigen Sie ungefähr 20 Minuten.

In diesem Tutorial erfahren Sie, wie Sie mithilfe von JavaScript Logik eine Lösung erstellen, die diesen Anwendungsfall ausführt. Sie erfahren beispielsweise, wie Sie eine Datenbank lesen, um zu bestimmen, welche Mitarbeiter das einjährige Jubiläum erreicht haben, wie Sie die Daten verarbeiten und mithilfe einer Lambda-Funktion eine Textnachricht senden. Anschließend erfahren Sie, wie Sie jeden Wochentag einen Cron-Ausdruck verwenden, um die Lambda-Funktion aufzurufen.

In diesem AWSTutorial wird eine Amazon-DynamoDB-Tabelle mit dem Namen Mitarbeiter verwendet, die diese Felder enthält.

- id – der Primärschlüssel für die Tabelle.
- firstName – Vorname des Mitarbeiters.
- Telefon – Telefonnummer des Mitarbeiters.
- startDate – Startdatum des Mitarbeiters.

<input type="checkbox"/>	Id <i>i</i>	first	phone	startDate
<input type="checkbox"/>	1	Scott	15555555654	2019-12-20
<input type="checkbox"/>	2	Malcolm	15555555654	2019-12-17
<input type="checkbox"/>	55	Lam	15555555654	2019-12-19

Important

Kosten für die Fertigstellung: Die in diesem Dokument enthaltenen AWS Services sind im AWS kostenlosen Kontingent für enthalten. Stellen Sie jedoch sicher, dass Sie alle Ressourcen beenden, nachdem Sie dieses Tutorial abgeschlossen haben, um sicherzustellen, dass Ihnen keine Kosten in Rechnung gestellt werden.

So erstellen Sie die App:

1. [Erfüllen der Voraussetzungen](#)
2. [Erstellen der AWS Ressourcen](#)
3. [Vorbereiten des Browserskripts](#)
4. [Erstellen und Hochladen einer Lambda-Funktion](#)
5. [Bereitstellen der Lambda-Funktion](#)
6. [Ausführen der App](#)
7. [Löschen der Ressourcen](#)

Voraussetzungen für Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese Node.js- TypeScript Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen unter [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer freigegebenen Anmeldeinformationsdatei finden Sie unter [Gemeinsam genutzte Konfigurations- und Anmeldeinformationsdateien](#) im AWS Referenzhandbuch für -SDKs und Tools.

Erstellen der AWS Ressourcen

Dieses Tutorial erfordert die folgenden Ressourcen.

- Eine Amazon-DynamoDB-Tabelle mit dem Namen Mitarbeiter mit einem Schlüssel mit dem Namen Id und den Feldern, die in der vorherigen Abbildung gezeigt wurden. Stellen Sie sicher, dass Sie

die richtigen Daten eingeben, einschließlich eines gültigen Mobiltelefons, mit dem Sie diesen Anwendungsfall testen möchten. Weitere Informationen finden Sie unter [Erstellen einer Tabelle](#).

- Eine IAM-Rolle mit angefügten Berechtigungen zum Ausführen von Lambda-Funktionen.
- Ein Amazon S3-Bucket zum Hosten der Lambda-Funktion.

Sie können diese Ressourcen manuell erstellen, aber wir empfehlen, diese Ressourcen mit der bereitzustellen, AWS CloudFormation wie in diesem Tutorial beschrieben.

Erstellen der AWS Ressourcen mit AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation-Benutzerhandbuch](#).

So erstellen Sie den AWS CloudFormationStack mit der AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI - Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier in GitHub](#) ihn.

Note

Die AWS CloudFormation Vorlage wurde mit der hier [auf GitHub](#) AWS CDK verfügbaren generiert. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

3. Führen Sie den folgenden Befehl über die Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

Important

Der Stack-Name muss innerhalb einer -AWSRegion und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben, und Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie im [AWS CLI - Befehlsreferenzhandbuch](#) und im [AWS CloudFormation -Benutzerhandbuch](#).

Zeigen Sie eine Liste der Ressourcen in der Konsole an, indem Sie den Stack im AWS CloudFormation Dashboard öffnen und die Registerkarte Ressourcen auswählen. Sie benötigen diese für das Tutorial.

4. Wenn der Stack erstellt wurde, verwenden Sie die , AWS SDK for JavaScript um die DynamoDB-Tabelle zu füllen, wie unter beschrieben [Füllen der DynamoDB-Tabelle](#).

Füllen der DynamoDB-Tabelle

Um die Tabelle zu füllen, erstellen Sie zunächst ein Verzeichnis mit dem Namen `libs`, und erstellen Sie darin eine Datei mit dem Namen `dynamoClient.js` und fügen Sie den folgenden Inhalt ein.

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );
// Set the AWS Region.
const REGION = "REGION"; // e.g. "us-east-1"
// Create an Amazon DynamoDB service client object.
const dynamoClient = new DynamoDBClient({region:REGION});
module.exports = { dynamoClient };
```

Dieser Code ist [hier auf GitHub](#) verfügbar.

Erstellen Sie als Nächstes eine Datei mit dem Namen `populate-table.js` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier in GitHub](#) diese Datei. Ersetzen Sie für eines der Elemente den Wert für die `-phone`Eigenschaft durch eine gültige Mobiltelefonnummer im E.164-Format und den Wert für die `startDate` durch das heutige Datum.

Führen Sie in der Befehlszeile den folgenden Befehl aus.

```
node populate-table.js
```

```
const {
BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
```

```
const {dynamoClient} = require( "./libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "1555555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "1555555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "1555555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

Dieser Code ist [hier auf GitHub](#) verfügbar.

Erstellen der AWS Lambda-Funktion

Konfigurieren des SDKs

Importieren Sie zunächst die erforderlichen AWS SDK for JavaScript (v3)-Module und -Befehle: `DynamoDBClient` und die `DynamoDB-ScanCommand` und `SNSClient` und den `AmazonSNS-PublishCommand` Befehl. Ersetzen Sie `REGION` durch die AWS Region. Berechnen Sie dann das heutige Datum und weisen Sie es einem Parameter zu. Erstellen Sie dann die Parameter für `ScanCommand`. Ersetzen Sie `TABLE_NAME` durch den Namen der Tabelle, die Sie im [Erstellen der AWS Ressourcen](#) Abschnitt dieses Beispiels erstellt haben.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündeln der Lambda-Funktion](#).)

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
```

```
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

Scannen der DynamoDB-Tabelle

Erstellen Sie zunächst eine Async/Await-Funktion mit dem Namen `sendText`, um eine Textnachricht mit dem Amazon SNS zu veröffentlichen `PublishCommand`. Fügen Sie dann ein `try` Blockmuster hinzu, das die DynamoDB-Tabelle nach Mitarbeitern mit ihrem heutigen Arbeitsjubiläum scannt und dann die `sendText` Funktion aufruft, um diesen Mitarbeitern eine Textnachricht zu senden. Wenn ein Fehler auftritt, wird der `catch` Block aufgerufen.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündeln der Lambda-Funktion](#).)

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!";
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
```



```
    console.log("Error, could not scan table ", err);
  }
};
```

Bündeln der Lambda-Funktion

In diesem Thema wird beschrieben, wie Sie die `myLambdafunction.js` und die erforderlichen AWS SDK for JavaScript Module für dieses Beispiel in einer gebündelten Datei namens `bündelnindex.js`.

1. Falls noch nicht geschehen, folgen Sie der [Voraussetzungen für Aufgaben](#) für dieses Beispiel, um Webpack zu installieren.

Note

Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#).

2. Führen Sie in der Befehlszeile Folgendes aus, um die JavaScript für dieses Beispiel in einer Datei namens `<index.js>` bündeln:

```
webpack mylambdafunction.js --mode development --target node --devtool false --output-library-target umd -o index.js
```

Important

Beachten Sie, dass die Ausgabe den Namen `hatindex.js`. Dies liegt daran, dass Lambda-Funktionen über einen `index.js` Handler verfügen müssen, um zu funktionieren.

3. Komprimieren Sie die gebündelte Ausgabedatei `index.js` in eine ZIP-Datei mit dem Namen `my-lambda-function.zip`.
4. Laden Sie `mylambdafunction.zip` in den Amazon S3-Bucket hoch, den Sie im [Erstellen der AWS Ressourcen](#) Thema dieses Tutorials erstellt haben.

Hier ist der vollständige Browserskriptcode für `mylambdafunction.js`.

```
"use strict";
```

```
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};

// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
```

```
const data = await dbclient.send(new ScanCommand(params));
data.Items.forEach(function (element, index, array) {
  const textParams = {
    PhoneNumber: element.phone.N,
    Message:
      "Hi " +
      element.firstName.S +
      "; congratulations on your work anniversary!",
  };
  // Send message using Amazon SNS.
  sendText(textParams);
});
} catch (err) {
  console.log("Error, could not scan table ", err);
}
};
```

Stellen Sie die Lambda-Funktion bereit

Erstellen Sie im Stammverzeichnis Ihres Projekts eine `lambda-function-setup.js` Datei und fügen Sie den unten stehenden Inhalt ein.

Ersetzen Sie *BUCKET_NAME* durch den Namen des Amazon S3-Buckets, in den Sie die ZIP-Version Ihrer Lambda-Funktion hochgeladen haben. Ersetzen Sie *ZIP_FILE_NAME* durch den Namen der ZIP-Version Ihrer Lambda-Funktion. Ersetzen Sie *IAM_ROLE_ARN* durch die Amazon-Ressourcennummer (ARN) der IAM-Rolle, die Sie im [Erstellen der AWS Ressourcen](#) Thema dieses Tutorials erstellt haben. Ersetzen Sie *LAMBDA_FUNCTION_NAME* durch einen Namen für die Lambda-Funktion.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
  lambdaClient
} = require("../libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
```

```
Code: {
  S3Bucket: "BUCKET_NAME", // BUCKET_NAME
  S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
},
FunctionName: "LAMBDA_FUNCTION_NAME",
Handler: "index.handler",
Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
lambda-tutorial-lambda-role
Runtime: "nodejs12.x",
Description:
  "Scans a DynamoDB table of employee details and using Amazon Simple Notification
  Services (Amazon SNS) to " +
  "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Geben Sie in der Befehlszeile Folgendes ein, um die Lambda-Funktion bereitzustellen.

```
node lambda-function-setup.js
```


Dieses Codebeispiel ist [hier auf GitHub](#) verfügbar.

Konfigurieren von CloudWatch zum Aufrufen der Lambda-Funktionen

So konfigurieren Sie CloudWatch , um die Lambda-Funktionen aufzurufen:

1. Öffnen Sie die Funktions-Seite in der Lambda-Konsole.
2. Wählen Sie die Lambda-Funktion.
3. Wählen Sie unter Designer die Option Add trigger (Trigger hinzufügen).
4. Legen Sie den Auslösertyp auf CloudWatch Events/EventBridge fest.
5. Wählen Sie für Regel die Option Neue Regel erstellen aus.

6. Geben Sie den Regelnamen und die Regelbeschreibung ein.
7. Wählen Sie für Regeltyp die Option Zeitplanausdruck aus.
8. Geben Sie im Feld Ausdruck planen einen Cron-Ausdruck ein. Zum Beispiel cron(0 12 ? * MON-FRI *)
9. Wählen Sie Hinzufügen aus.

 Note

Weitere Informationen finden Sie unter [Verwenden von Lambda mit CloudWatch Ereignissen](#).

Löschen der Ressourcen

Herzlichen Glückwunsch! Sie haben eine Lambda-Funktion über von Amazon CloudWatch geplante Ereignisse mit der aufgerufenen AWS SDK for JavaScript. Wie zu Beginn dieses Tutorials erwähnt, sollten Sie alle Ressourcen beenden, die Sie während dieses Tutorials erstellen, um sicherzustellen, dass Ihnen keine Kosten in Rechnung gestellt werden. Sie können dies tun, indem Sie den AWS CloudFormationStack, den Sie im [Erstellen der AWS Ressourcen](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

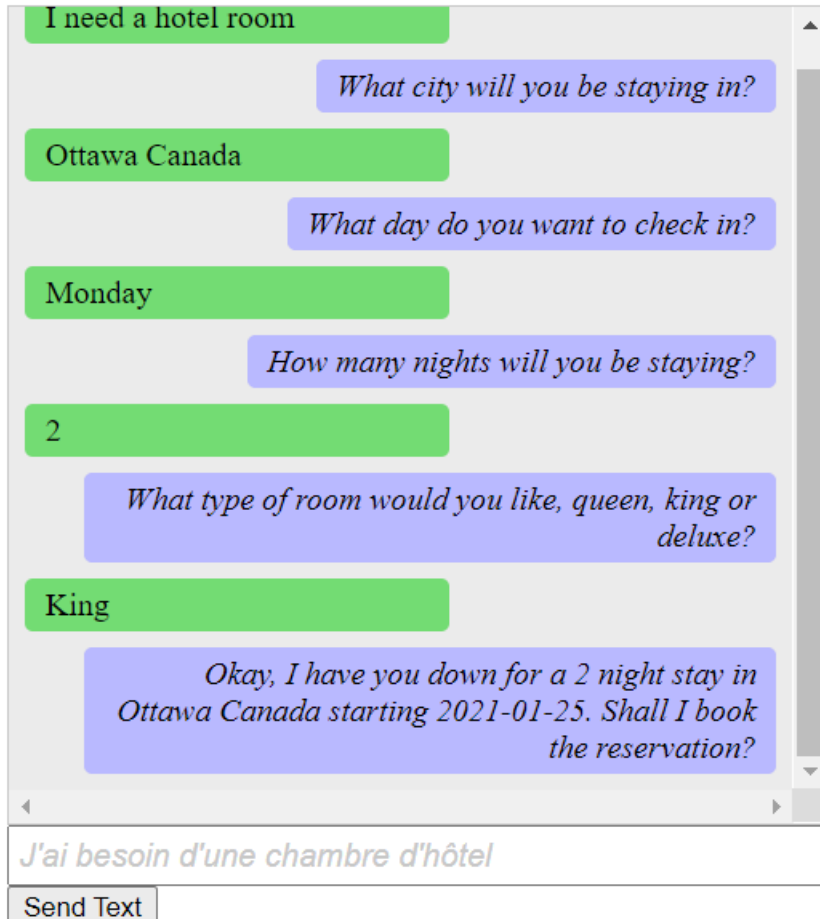
1. Öffnen Sie die [AWS CloudFormation-Konsole](#).
2. Wählen Sie auf der Seite Stacks den Stack aus.
3. Wählen Sie Löschen.

Erstellen eines Amazon Lex-Chatbots

Sie können einen Amazon Lex-Chatbot innerhalb einer Webanwendung erstellen, um Ihre Website-Besucher einzubeziehen. Ein Amazon Lex-Chatbot ist eine Funktion, die Online-Chat-Konversationen mit Benutzern durchführt, ohne direkten Kontakt mit einer Person herzustellen. Die folgende Abbildung zeigt beispielsweise einen Amazon Lex-Chatbot, der einen Benutzer über die Reservierung eines Parkraums informiert.

Amazon Lex - BookTrip

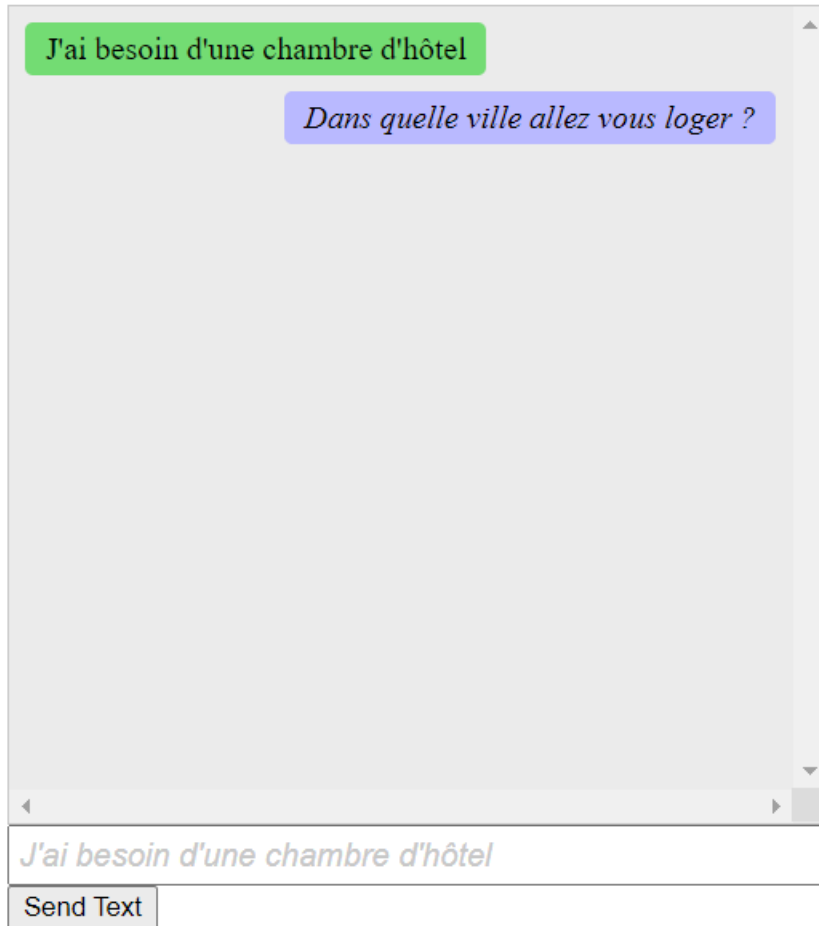
This multiple language chatbot shows you how easy it is to incorporate [Amazon Lex](#) into your web apps. Try it out.



Der in diesem AWSTutorial erstellte Amazon Lex-Chatbot kann mehrere Sprachen verarbeiten. Ein Benutzer, der Französisch spricht, kann beispielsweise Französischen Text eingeben und eine Antwort auf Französisch zurückgeben.

Amazon Lex - BookTrip

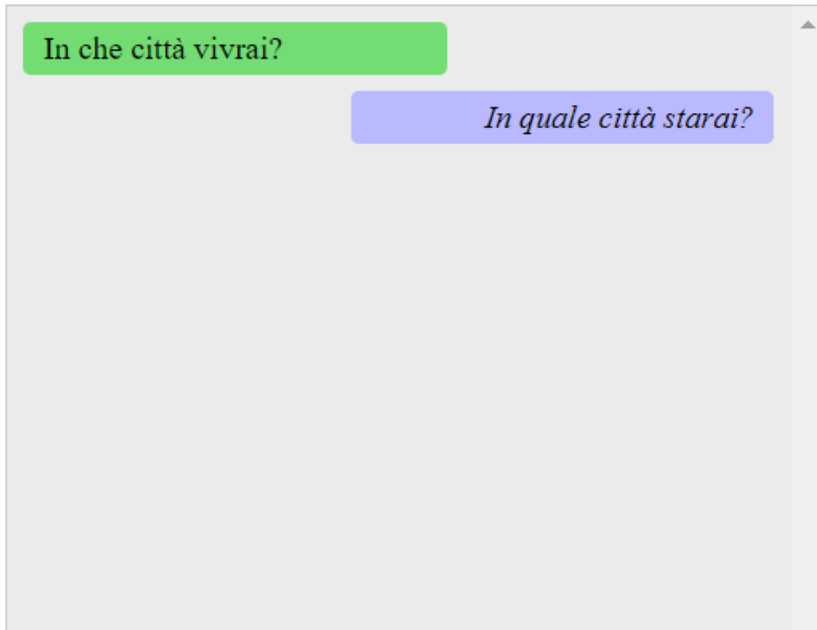
This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Ebenso kann ein Benutzer mit dem Amazon Lex-Chatbot auf Italienisch kommunizieren.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Dieses AWSTutorial führt Sie durch das Erstellen eines Amazon Lex-Chatbots und dessen Integration in eine Node.js-Webanwendung. Die AWS SDK for JavaScript (v3) wird verwendet, um diese AWS Services aufzurufen:

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Kosten für die Fertigstellung: Die in diesem Dokument enthaltenen AWS Services sind im [AWS kostenlosen Kontingent](#) für enthalten.

Hinweis: Stellen Sie sicher, dass Sie alle Ressourcen beenden, die Sie während dieses Tutorials erstellen, um sicherzustellen, dass Ihnen keine Kosten in Rechnung gestellt werden.

So erstellen Sie die App:

1. [Voraussetzungen](#)
2. [Bereitstellen von Ressourcen](#)

3. [Amazon Lex-Chatbot erstellen](#)
4. [Erstellen des HTML](#)
5. [Erstellen des Browserskripts](#)
6. [Nächste Schritte](#)

Voraussetzungen

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Knotenbeispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen unter [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer freigegebenen Anmeldeinformationsdatei finden Sie unter [Gemeinsam genutzte Konfigurations- und Anmeldeinformationsdateien](#) im AWS Referenzhandbuch für -SDKs und Tools.

Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js-Downloads](#).

Wenn Sie jedoch lieber die CommonJS-Syntax verwenden möchten, lesen Sie bitte [JavaScript ES6/CommonJS-Syntax](#).

Erstellen der AWS Ressourcen

Dieses Tutorial erfordert die folgenden Ressourcen.

- Eine nicht authentifizierte IAM-Rolle mit angefügten Berechtigungen für:
 - Amazon Comprehend
 - Amazon Translate
 - Amazon Lex

Sie können diese Ressourcen manuell erstellen, aber wir empfehlen, diese Ressourcen mit bereitzustellen, AWS CloudFormation wie in diesem Tutorial beschrieben.

Erstellen der AWS Ressourcen mit AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation-Benutzerhandbuch](#).

So erstellen Sie den AWS CloudFormationStack mit der AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI - Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier in GitHub](#) diese Datei.

Note

Die AWS CloudFormation Vorlage wurde mit der hier [auf GitHub](#) AWS CDK verfügbaren generiert. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

3. Führen Sie den folgenden Befehl über die Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

Important

Der Stack-Name muss innerhalb einer -AWSRegion und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben, und Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie im [AWS CLI - Befehlsreferenzhandbuch](#) und im [AWS CloudFormation -Benutzerhandbuch](#).

Um die erstellten Ressourcen anzuzeigen, öffnen Sie die Amazon Lex-Konsole, wählen Sie den Stack und wählen Sie die Registerkarte Ressourcen aus.

Erstellen Sie einen Amazon-Lex-Bot

⚠ Important

Verwenden Sie V1 der Amazon Lex-Konsole, um den Bot zu erstellen. Dieses Beispiel funktioniert nicht mit Bots, die mit V2 erstellt wurden.

Der erste Schritt besteht darin, einen Amazon Lex-Chatbot mithilfe der Amazon-Web-Services-Managementkonsole zu erstellen. In diesem Beispiel wird das Amazon Lex-BookTripBeispiel verwendet. Weitere Informationen finden Sie unter [Reise buche](#).

- Melden Sie sich bei der Amazon Web Services Management Console an und öffnen Sie die Amazon Lex-Konsole unter [Amazon Web Services Console](#).
- Wählen Sie auf der Seite Bots die Option Erstellen aus.
- Wählen Sie BookTrip Blueprint (überlassen Sie den Standard-Bot-Namen BookTrip).

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.

CREATE YOUR OWN TRY A SAMPLE

Custom bot **BookTrip** OrderFlowers ScheduleAppointment

Bot name

BookTrip



- Geben Sie die Standardeinstellungen ein und wählen Sie Erstellen (die Konsole zeigt den BookTrip Bot an). Überprüfen Sie auf der Registerkarte Editor die Details der vorkonfigurierten Absichten.
- Testen Sie den Bot im Testfenster. Beginnen Sie den Test, indem Sie Ich möchte ein Parkhaus reservieren eingeben.

> Test bot (Latest) ✔ Ready. Build complete

I want to book a hotel room

What city will you be staying in?

Clear chat history

 Chat with your bot...

Inspect response

Dialog State: ElicitSlot

[Hide](#)

Summary Detail

Intent: BookHotel

- Wählen Sie Veröffentlichen und geben Sie einen Aliasnamen an (diesen Wert benötigen Sie, wenn Sie die verwenden AWS SDK for JavaScript).

Note

Sie müssen in Ihrem JavaScript Code auf den Bot-Namen und den Bot-Alias verweisen.

Erstellen des HTML

Erstellen Sie eine Datei namens `index.html`. Kopieren Sie den folgenden Code in und fügen Sie ihn in `index.html`. Dieser HTML-Code verweist auf `main.js`. Dies ist eine gebündelte Version von `index.js`, die die erforderlichen AWS SDK for JavaScript Module enthält. Sie erstellen diese Datei

in [Erstellen des HTML](#). verweist `index.html` auch auf `style.css`, wodurch die Stile hinzugefügt werden.

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
  <link type="text/css" rel="stylesheet" href="style.css" />
</head>

<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
    >Amazon Lex</a>
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

Dieser Code ist auch [hier auf GitHub](#) verfügbar.

Erstellen des Browserskripts

Erstellen Sie eine Datei namens `index.js`. Kopieren Sie den folgenden Code und fügen Sie ihn in ein `index.js`. Importieren Sie die erforderlichen AWS SDK for JavaScript Module und Befehle. Erstellen Sie Clients für Amazon Lex, Amazon Comprehend und Amazon Translate. Ersetzen Sie **REGION** durch AWS Region und **IDENTITY_POOL_ID** durch die ID des Identitätspools, den Sie in

der erstellt haben [Erstellen der AWS Ressourcen](#) . Um diese Identitätspool-ID abzurufen, öffnen Sie den Identitätspool in der Amazon Cognito-Konsole, wählen Sie Identitätspool bearbeiten und wählen Sie im Seitenmenü Beispielcode aus. Die Identitätspool-ID wird in der Konsole in Rottext angezeigt.

Erstellen Sie zunächst ein `libs` Verzeichnis, um die erforderlichen Service-Client-Objekte zu erstellen `comprehendClient.js`, indem Sie drei Dateien erstellen, `lexClient.js`, und `translateClient.js`. Fügen Sie den entsprechenden Code unten in jede Datei ein und ersetzen Sie `REGION` und `IDENTITY_POOL_ID` in jeder Datei.

Note

Verwenden Sie die ID des Amazon Cognito-Identitätspools, den Sie in [Erstellen der AWS Ressourcen mit AWS CloudFormation](#).

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { ComprehendClient } from "@aws-sdk/client-comprehend";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const comprehendClient = new ComprehendClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.
```

```
// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { lexClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { TranslateClient } from "@aws-sdk/client-translate";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Translate service client object.
const translateClient = new TranslateClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { translateClient };
```

Dieser Code ist [hier auf verfügbar GitHub](#).

Erstellen Sie als Nächstes eine `-index.js` Datei und fügen Sie den folgenden Code ein.

Ersetzen Sie `_ALIAS` und `_NAME` durch den Alias bzw. den Namen Ihres Amazon Lex-Bots bzw. `USER_ID` durch eine Benutzer-ID. Die `createResponse` asynchrone Funktion führt Folgendes aus:

- Nimmt den vom Benutzer eingegebenen Text in den Browser und verwendet Amazon Comprehend, um seinen Sprachcode zu bestimmen.
- Nimmt den Sprachcode und verwendet Amazon Translate, um den Text ins Englische zu übersetzen.

- Nimmt den übersetzten Text und verwendet Amazon Lex, um eine Antwort zu generieren.
- Veröffentlicht die Antwort auf der Browserseite.

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "../libs/lexClient.js";
import { translateClient } from "../libs/translateClient.js";
import { comprehendClient } from "../libs/comprehendClient.js";

var g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  var conversationDiv = document.getElementById("conversation");
  var requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  var conversationDiv = document.getElementById("conversation");
  var responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  var lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
```



```
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
xhr.send("text=" + text);
}

function loadNewItems() {
  showRequest();

  // Re-enable input.
  var wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}

// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  var wisdomText = document.getElementById("wisdom");
  if (wisdomText && wisdomText.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    var wisdom = wisdomText.value.trim();
    wisdomText.value = "...";
    wisdomText.locked = true;
    handleText(wisdom);


    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams)
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams)
        );
      }
    }
  }
}
```

```
);
console.log("Success. Translated text: ", data.TranslatedText);
const lexParams = {
  botName: "BookTrip",
  botAlias: "mynewalias",
  inputText: data.TranslatedText,
  userId: "chatbot", // For example, 'chatbot-demo'.
};
try {
  const data = await lexClient.send(new PostTextCommand(lexParams));
  console.log("Success. Response is: ", data.message);
  var msg = data.message;
  showResponse(msg);
} catch (err) {
  console.log("Error responding to message. ", err);
}
} catch (err) {
  console.log("Error translating text. ", err);
}
} catch (err) {
  console.log("Error identifying language. ", err);
}
}
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

Dieser Code ist [hier auf verfügbar GitHub](#).

Verwenden Sie jetzt Webpack, um die AWS SDK for JavaScript Module `index.js` und in einer einzigen Datei zu bündeln, `main.js`.

1. Falls noch nicht geschehen, folgen Sie der [Voraussetzungen](#) für dieses Beispiel, um Webpack zu installieren.

 Note

Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#).

2. Führen Sie in der Befehlszeile Folgendes aus, um die JavaScript für dieses Beispiel in einer Datei namens zu bündeln `main.js`:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Nächste Schritte

Herzlichen Glückwunsch! Sie haben eine Node.js-Anwendung erstellt, die Amazon Lex verwendet, um ein interaktives Benutzererlebnis zu schaffen. Wie zu Beginn dieses Tutorials erwähnt, sollten Sie alle Ressourcen beenden, die Sie während dieses Tutorials erstellen, um sicherzustellen, dass Ihnen keine Kosten in Rechnung gestellt werden. Sie können dies tun, indem Sie den AWS CloudFormationStack, den Sie im [Erstellen der AWS Ressourcen](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie die [AWS CloudFormation-Konsole](#).
2. Wählen Sie auf der Seite Stacks den Stack aus.
3. Wählen Sie Löschen aus.

Weitere AWS serviceübergreifende Beispiele finden Sie unter [AWS SDK for JavaScript serviceübergreifende Beispiele](#).

Erstellen einer Beispiel-Messaging-Anwendung

Sie können eine AWS Anwendung erstellen, die Nachrichten sendet und abrufen, indem Sie die AWS SDK for JavaScript und Amazon Simple Queue Service (Amazon SQS) verwenden. Nachrichten werden in einer FIFO-Warteschlange (First in, First out) gespeichert, die sicherstellt, dass die Reihenfolge der Nachrichten konsistent ist. Die erste Nachricht, die in der Warteschlange gespeichert ist, ist beispielsweise die erste Nachricht, die aus der Warteschlange gelesen wird.

Note

Weitere Informationen zu Amazon SQS finden Sie unter [Was ist Amazon Simple Queue Service?](#)

In diesem Tutorial erstellen Sie eine Node.js-Anwendung mit dem Namen AWS Messaging.

Kosten für die Fertigstellung: Die in diesem Dokument enthaltenen AWS Services sind im [AWS kostenlosen Kontingent](#) für enthalten.

Hinweis: Stellen Sie sicher, dass Sie alle Ressourcen beenden, die Sie während dieses Tutorials erstellen, um sicherzustellen, dass Ihnen keine Kosten entstehen.

So erstellen Sie die App:

1. [Voraussetzungen](#)
2. [Bereitstellen von Ressourcen](#)
3. [Den Workflow verstehen](#)
4. [Erstellen des HTML](#)
5. [Erstellen des Browserskripts](#)
6. [Nächste Schritte](#)

Voraussetzungen

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Knotenbeispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK for JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen unter [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zum Bereitstellen einer freigegebenen Anmeldeinformationsdatei finden Sie unter [Gemeinsam genutzte Konfigurations- und Anmeldeinformationsdateien](#) im AWS Referenzhandbuch für -SDKs und Tools.

Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js-Downloads](#).

Wenn Sie jedoch lieber die CommonJS-Syntax verwenden möchten, lesen Sie bitte [JavaScript ES6/CommonJS-Syntax](#).

Erstellen der AWS Ressourcen

Dieses Tutorial erfordert die folgenden Ressourcen.

- Eine nicht authentifizierte IAM-Rolle mit Berechtigungen für Amazon SQS .
- Eine FIFOAmazon SQS-Warteschlange mit dem Namen Message.fifo – Informationen zum Erstellen einer Warteschlange finden Sie unter [Erstellen einer Amazon SQS-Warteschlange](#).

Sie können diese Ressourcen manuell erstellen, aber wir empfehlen, diese Ressourcen mithilfe der AWS CloudFormation (AWS CloudFormation) bereitzustellen, wie in diesem Tutorial beschrieben.

Note

ist AWS CloudFormation ein Softwareentwicklungs-Framework, mit dem Sie Cloud-Anwendungsressourcen definieren können. Weitere Informationen finden Sie im [AWS CloudFormation-Benutzerhandbuch](#).

Erstellen der AWS Ressourcen mithilfe der AWS CloudFormation

AWS CloudFormation ermöglicht es Ihnen, die AWS-Infrastrukturen vorhersagbar und wiederholt zu erstellen und bereitzustellen. Weitere Informationen zu AWS CloudFormation finden Sie im [AWS CloudFormation-Benutzerhandbuch](#).

So erstellen Sie den AWS CloudFormationStack mit der AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI - Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier in GitHub](#) diese Datei.

Note

Die AWS CloudFormation Vorlage wurde mit der hier [auf GitHub](#) AWS CDK verfügbaren generiert. Weitere Informationen zum AWS CDK finden Sie im [AWS Cloud Development Kit \(AWS CDK\)Entwicklerhandbuch für](#).

3. Führen Sie den folgenden Befehl über die Befehlszeile aus und ersetzen Sie `STACK_NAME` durch einen eindeutigen Namen für den Stack.

Important

Der Stack-Name muss innerhalb einer -AWSRegion und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben, und Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie im [AWS CLI - Befehlsreferenzhandbuch](#) und im [AWS CloudFormation -Benutzerhandbuch](#).

Um die erstellten Ressourcen anzuzeigen, öffnen Sie AWS CloudFormation in der -AWSManagementkonsole, wählen Sie den Stack und wählen Sie die Registerkarte Ressourcen aus.

Verstehen der AWS Messaging-Anwendung

Um eine Nachricht an eine SQS-Warteschlange zu senden, geben Sie die Nachricht in die Anwendung ein und wählen Sie Senden.

Nachdem die Nachricht gesendet wurde, zeigt die Anwendung die Nachricht an.

Sie können Bereinigen wählen, um die Nachrichten aus der Amazon SQS-Warteschlange zu löschen. Dies führt zu einer leeren Warteschlange und es werden keine Nachrichten in der Anwendung angezeigt.

Im Folgenden wird beschrieben, wie die Anwendung eine Nachricht verarbeitet:

- Der Benutzer wählt seinen Namen aus, gibt seine Nachricht ein und sendet die Nachricht, die die `pushMessage` Funktion initiiert.
- `pushMessage` ruft die Amazon SQS-Warteschlangen-URL ab und sendet dann eine Nachricht mit einem eindeutigen Nachrichten-ID-Wert (eine GUID) dem Nachrichtentext und dem Benutzer an die Amazon SQS-Warteschlange.
- `pushMessage` ruft die Nachrichten aus der Amazon SQS-Warteschlange ab, extrahiert den Benutzer und die Nachricht für jede Nachricht und zeigt die Nachrichten an.

- Der Benutzer kann die Nachrichten löschen, die die Nachrichten aus der Amazon SQS-Warteschlange und aus der Benutzeroberfläche löschen.

Erstellen der HTML-Seite

Jetzt erstellen Sie die HTML-Dateien, die für die grafische Benutzeroberfläche (GUI) der Anwendung erforderlich sind. Erstellen Sie eine Datei namens `index.html`. Kopieren Sie den folgenden Code in und fügen Sie ihn in `index.html`. Dieser HTML-Code verweist auf `main.js`. Dies ist eine gebündelte Version von `index.js`, die die erforderlichen AWS SDK for JavaScript Module enthält.

```
<!doctype html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="./images/favicon.ico" />
    <link
      rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    />
    <link rel="stylesheet" href="./css/styles.css" />
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
    <script src="./js/main.js"></script>
    <style>
      .messageelement {
        margin: auto;
        border: 2px solid #dedede;
        background-color: #d7d1d0;
        border-radius: 5px;
        max-width: 800px;
        padding: 10px;
        margin: 10px 0;
      }

      .messageelement::after {
        content: "";
        clear: both;
      }
    </style>
  </head>
</html>
```

```
    display: table;
  }

  .messageelement img {
    float: left;
    max-width: 60px;
    width: 100%;
    margin-right: 20px;
    border-radius: 50%;
  }

  .messageelement img.right {
    float: right;
    margin-left: 20px;
    margin-right: 0;
  }
</style>
</head>
<body>
  <div class="container">
    <h2>AWS Sample Messaging Application</h2>
    <div id="messages"></div>

    <div class="input-group mb-3">
      <div class="input-group-prepend">
        <span class="input-group-text" id="basic-addon1">Sender:</span>
      </div>
      <select name="cars" id="username">
        <option value="Scott">Brian</option>
        <option value="Tricia">Tricia</option>
      </select>
    </div>

    <div class="input-group">
      <div class="input-group-prepend">
        <span class="input-group-text">Message:</span>
      </div>
      <textarea
        class="form-control"
        id="textarea"
        aria-label="With textarea"
      ></textarea>
      <button
        type="button"
```



```

        onclick="pushMessage()"
        id="send"
        class="btn btn-success"
    >
    Send
</button>
<button
    type="button"
    onclick="purge()"
    id="refresh"
    class="btn btn-success"
    >
    Purge
</button>
</div>
<!-- All of these child items are hidden and only displayed in a FancyBox
----->
<div id="hide" style="display: none">
    <div id="base" class="messageelement">
        
        <p id="text">Excellent! So, what do you want to do today?</p>
        <span class="time-right">11:02</span>
    </div>
</div>
</div>
</body>
</html>

```

Dieser Code ist auch [hier auf GitHub](#) verfügbar.

Erstellen des Browserskripts

In diesem Thema erstellen Sie ein Browserskript für die App. Wenn Sie das Browserskript erstellt haben, bündeln Sie es in einer Datei namens `main.js` wie unter beschrieben [Bündeln der JavaScript](#).

Erstellen Sie eine Datei namens `index.js`. Kopieren Sie den Code von [hier aus GitHub](#) und fügen Sie ihn ein.

Dieser Code wird in den folgenden Abschnitten erläutert:

1. [Konfiguration](#)
2. [populateChat](#)
3. [Push-Nachrichten](#)
4. [löschen](#)

Konfiguration

Erstellen Sie zunächst ein `libs` Verzeichnis, um das erforderliche Amazon SQS-Clientobjekt zu erstellen, indem Sie Dateien mit dem Namen `erstellensqsClient.js`. Ersetzen Sie jeweils **REGION** und **IDENTITY_POOL_ID**.

Note

Verwenden Sie die ID des Amazon Cognito-Identitätspools, den Sie in [erstellt haben](#) [Erstellen der AWS Ressourcen](#) .

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { SQSClient } from "@aws-sdk/client-sqs";
const REGION = "REGION"; //e.g. "us-east-1"
const IdentityPoolId = "IDENTITY_POOL_ID";
const sqsClient = new SQSClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IdentityPoolId
  }),
});
```

`index.js` Importieren Sie in der die erforderlichen AWS SDK for JavaScript Module und Befehle. Ersetzen Sie **SQS_QUEUE_NAME** durch den Namen der Amazon SQS-Warteschlange, die Sie in der erstellt haben [Erstellen der AWS Ressourcen](#) .

```
import {
```

```
GetQueueUrlCommand,  
SendMessageCommand,  
ReceiveMessageCommand,  
PurgeQueueCommand,  
} from "@aws-sdk/client-sqs";  
import { sqsClient } from "../libs/sqsClient.js";  
  
const QueueName = "SQS_QUEUE_NAME"; // The Amazon SQS queue name, which must end  
in .fifo for this example.
```

populateChat

Das Laden der `populateChat` Funktion ruft automatisch die URL für die Amazon SQS-Warteschlange ab, ruft alle Nachrichten in der Warteschlange ab und zeigt sie an.

```
$(function () {  
  populateChat();  
});  
  
const populateChat = async () => {  
  try {  
    // Set the Amazon SQS Queue parameters.  
    const queueParams = {  
      QueueName: QueueName,  
      Attributes: {  
        DelaySeconds: "60",  
        MessageRetentionPeriod: "86400",  
      },  
    };  
    // Get the Amazon SQS Queue URL.  
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));  
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);  
    // Set the parameters for retrieving the messages in the Amazon SQS Queue.  
    var getMessageParams = {  
      QueueUrl: data.QueueUrl,  
      MaxNumberOfMessages: 10,  
      MessageAttributeNames: ["All"],  
      VisibilityTimeout: 20,  
      WaitTimeSeconds: 20,  
    };  
    try {  
      // Retrieve the messages from the Amazon SQS Queue.
```

```
const data = await sqsClient.send(
  new ReceiveMessageCommand(getMessageParams)
);
console.log("Successfully retrieved messages", data.Messages);

// Loop through messages for user and message body.
var i;
for (i = 0; i < data.Messages.length; i++) {
  const name = data.Messages[i].MessageAttributes.Name.StringValue;
  const body = data.Messages[i].Body;
  // Create the HTML for the message.
  var userText = body + "<br><br><b>" + name;
  var myTextNode = $("#base").clone();
  myTextNode.text(userText);
  var image_url;
  var n = name.localeCompare("Scott");
  if (n == 0) image_url = "./images/av1.png";
  else image_url = "./images/av2.png";
  var images_div =
    '';
  myTextNode.html(userText);
  myTextNode.append(images_div);

  // Add the message to the GUI.
  $("#messages").append(myTextNode);
}
} catch (err) {
  console.log("Error loading messages: ", err);
}
} catch (err) {
  console.log("Error retrieving SQS queue URL: ", err);
}
};
```

Push-Nachrichten

Der Benutzer wählt seinen Namen aus und gibt seine Nachricht ein und sendet die Nachricht, die die `pushMessage` Funktion initiiert. `pushMessage` ruft die Amazon SQS-Warteschlangen-URL ab und sendet dann eine Nachricht mit einem eindeutigen Nachrichten-ID-Wert (eine GUID) den Nachrichtentext und den Benutzer an die Amazon SQS-Warteschlange. Anschließend werden alle Nachrichten aus der Amazon SQS-Warteschlange abgerufen und angezeigt.

```
const pushMessage = async () => {
  // Get and convert user and message input.
  var user = $("#username").val();
  var message = $("#textarea").val();

  // Create random deduplication ID.
  var dt = new Date().getTime();
  var uuid = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
    c
  ) {
    var r = (dt + Math.random() * 16) % 16 | 0;
    dt = Math.floor(dt / 16);
    return (c == "x" ? r : (r & 0x3) | 0x8).toString(16);
  });

  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success. The URL of the SQS Queue is: ", data.QueueUrl);
    // Set the parameters for the message.
    var messageParams = {
      MessageAttributes: {
        Name: {
          DataType: "String",
          StringValue: user,
        },
      },
      MessageBody: message,
      MessageDeduplicationId: uuid,
      MessageGroupId: "GroupA",
      QueueUrl: data.QueueUrl,
    };
    const result = await sqsClient.send(new SendMessageCommand(messageParams));
    console.log("Success", result.MessageId);

    // Set the parameters for retrieving all messages in the SQS queue.
```

```
var getMessageParams = {
  QueueUrl: data.QueueUrl,
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  VisibilityTimeout: 20,
  WaitTimeSeconds: 20,
};

// Retrieve messages from SQS Queue.
const final = await sqsClient.send(
  new ReceiveMessageCommand(getMessageParams)
);
console.log("Successfully retrieved", final.Messages);
$("#messages").empty();
// Loop through messages for user and message body.
var i;
for (i = 0; i < final.Messages.length; i++) {
  const name = final.Messages[i].MessageAttributes.Name.StringValue;
  const body = final.Messages[i].Body;
  // Create the HTML for the message.
  var userText = body + "<br><br><b>" + name;
  var myTextNode = $("#base").clone();
  myTextNode.text(userText);
  var image_url;
  var n = name.localeCompare("Scott");
  if (n == 0) image_url = "./images/av1.png";
  else image_url = "./images/av2.png";
  var images_div =
    '';
  myTextNode.html(userText);
  myTextNode.append(images_div);
  // Add the HTML to the GUI.
  $("#messages").append(myTextNode);
}
} catch (err) {
  console.log("Error", err);
}
};
// Make the function available to the browser window.
window.pushMessage = pushMessage;
```

Löschen von Nachrichten

`purge` löscht die Nachrichten aus der Amazon SQS-Warteschlange und aus der Benutzeroberfläche.

```
// Delete the message from the Amazon SQS queue.
const purge = async () => {
  try {
    // Set the Amazon SQS Queue parameters.
    const queueParams = {
      QueueName: QueueName,
      Attributes: {
        DelaySeconds: "60",
        MessageRetentionPeriod: "86400",
      },
    };
    // Get the Amazon SQS Queue URL.
    const data = await sqsClient.send(new GetQueueUrlCommand(queueParams));
    console.log("Success", data.QueueUrl);
    // Delete all the messages in the Amazon SQS Queue.
    const result = await sqsClient.send(
      new PurgeQueueCommand({ QueueUrl: data.QueueUrl })
    );
    // Delete all the messages from the GUI.
    $("#messages").empty();
    console.log("Success. All messages deleted.", data);
  } catch (err) {
    console.log("Error", err);
  }
};

// Make the function available to the browser window.
window.purge = purge;
```

Bündeln der JavaScript

Dieser vollständige Browserskriptcode ist [hier auf verfügbar GitHub](#).

Verwenden Sie jetzt Webpack, um die AWS SDK for JavaScript Module `index.js` und in einer einzigen Datei zu bündeln, `main.js`.

1. Falls noch nicht geschehen, folgen Sie der [Voraussetzungen](#) für dieses Beispiel, um Webpack zu installieren.

Note

Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#).

2. Führen Sie in der Befehlszeile Folgendes aus, um die JavaScript für dieses Beispiel in einer Datei namens zu `<index.js>` bündeln:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Nächste Schritte

Herzlichen Glückwunsch! Sie haben die AWS Messaging-Anwendung erstellt und bereitgestellt, die Amazon SQS verwendet. Wie zu Beginn dieses Tutorials erwähnt, sollten Sie alle Ressourcen beenden, die Sie während dieses Tutorials erstellen, um sicherzustellen, dass Ihnen keine weiteren Kosten mehr in Rechnung gestellt werden. Sie können dies tun, indem Sie den AWS CloudFormationStack, den Sie im [Erstellen der AWS Ressourcen](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie die [AWS CloudFormation in der -AWSManagementkonsole](#).
2. Öffnen Sie die Seite Stacks und wählen Sie den Stack aus.
3. Wählen Sie Löschen.

Verwenden von AWS Cloud9 mit der AWS SDK for JavaScript

Sie können AWS Cloud9 mit der verwenden, AWS SDK for JavaScript um Ihre JavaScript im Browsercode zu schreiben und auszuführen – sowie Ihren Node.js-Code zu schreiben, auszuführen und zu debuggen – und zwar nur über einen Browser. AWS Cloud9 enthält Tools wie einen Code-Editor und ein Terminal sowie einen Debugger für Node.js-Code.

Da die AWS Cloud9 IDE cloudbasiert ist, können Sie von Ihrem Büro, Ihrem Haus oder von überall aus mit einem mit dem Internet verbundenen Computer an Ihren Projekten arbeiten. Allgemeine Informationen zu AWS Cloud9 finden Sie im [AWS Cloud9 -Benutzerhandbuch](#).

In den folgenden Schritten wird beschrieben, wie Sie AWS Cloud9 mit dem SDK für einrichten JavaScript.

Inhalt

- [Schritt 1: Einrichten Ihres - AWS Kontos zur Verwendung von AWS Cloud9](#)
- [Schritt 2: Einrichten Ihrer AWS Cloud9 Entwicklungsumgebung](#)
- [Schritt 3: Einrichten des SDK für JavaScript](#)
 - [So richten Sie das SDK für JavaScript für Node.js ein](#)
 - [So richten Sie das SDK für JavaScript im Browser ein](#)
- [Schritt 4: Beispielcode herunterladen](#)
- [Schritt 5: Beispielcode ausführen und debuggen](#)

Schritt 1: Einrichten Ihres - AWS Kontos zur Verwendung von AWS Cloud9

Beginnen Sie mit der Verwendung von , AWS Cloud9 indem Sie sich bei der - AWS Cloud9 Konsole als AWS Identity and Access Management (IAM)-Entität (z. B. als IAM-Benutzer) anmelden, die über Zugriffsberechtigungen für AWS Cloud9 in Ihrem AWS Konto verfügt.

Informationen zum Einrichten einer IAM-Entität in Ihrem AWS Konto für den Zugriff auf AWS Cloud9 und zum Anmelden bei der AWS Cloud9 Konsole finden Sie unter [Teameinrichtung für AWS Cloud9](#) im AWS Cloud9 -Benutzerhandbuch.

Schritt 2: Einrichten Ihrer AWS Cloud9 Entwicklungsumgebung

Nachdem Sie sich bei der - AWS Cloud9 Konsole angemeldet haben, verwenden Sie die -Konsole, um eine - AWS Cloud9 Entwicklungsumgebung zu erstellen. Nachdem Sie die Umgebung erstellt haben, AWS Cloud9 öffnet die IDE für diese Umgebung.

Weitere Informationen finden Sie unter [Erstellen einer Umgebung in AWS Cloud9](#) im AWS Cloud9 - Benutzerhandbuch.

Note

Wenn Sie Ihre Umgebung in der Konsole zum ersten Mal erstellen, empfehlen wir, dass Sie die Option zum Erstellen einer neuen Instance für die Umgebung (EC2) verwenden. Diese Option weist an, eine Umgebung AWS Cloud9 zu erstellen, eine Amazon EC2-Instance zu starten und dann die neue Instance mit der neuen Umgebung zu verbinden. Dies ist der schnellste Weg, um mit der Verwendung von zu beginnen AWS Cloud9.

Schritt 3: Einrichten des SDK für JavaScript

Nachdem die IDE für Ihre Entwicklungsumgebung AWS Cloud9 geöffnet hat, befolgen Sie eines oder beide der folgenden Verfahren, um die IDE zum Einrichten des SDK für JavaScript in Ihrer Umgebung zu verwenden.

So richten Sie das SDK für JavaScript für Node.js ein

1. Wenn das Terminal in der IDE noch nicht geöffnet ist, öffnen Sie es. Wählen Sie in diesem Fall auf der Menüleiste in der IDE Window, New Terminal (Fenster, Neues Terminal) aus.
2. Führen Sie den folgenden Befehl aus, npm um mit den C1oud9 Client des SDK for zu installieren JavaScript.

```
npm install @aws-sdk/client-cloud9
```

Wenn die IDE nicht finden kann npm, führen Sie nacheinander die folgenden Befehle in der folgenden Reihenfolge aus, um zu installieren npm. (Diese Befehle setzen voraus, dass Sie die Option zum Erstellen einer neuen Instance für die Umgebung (EC2) ausgewählt haben, wie weiter oben in diesem Thema beschrieben.)

⚠ Warning

AWS steuert nicht den folgenden Code. Bevor Sie ihn ausführen, überprüfen Sie unbedingt dessen Authentizität und Integrität. Weitere Informationen zu diesem Code finden Sie im [nvm](#) (Node Version Manager) GitHub -Repository.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #  
Download and install Node Version Manager (nvm).  
. ~/.bashrc #  
Activate nvm.  
nvm install node #  
Use nvm to install npm (and Node.js at the same time).
```

So richten Sie das SDK für JavaScript im Browser ein

Um das SDK für JavaScript auf Ihren HTML-Seiten zu verwenden, verwenden Sie `Webpack` um die erforderlichen Clientmodule und alle erforderlichen JavaScript Funktionen in einer einzigen JavaScript Datei zu bündeln, und fügen Sie es in einem Skript-Tag in der `<head>` Ihrer HTML-Seiten hinzu. Beispielsweise:

```
<script src=./main.js></script>
```

i Note

Weitere Informationen zu Webpack finden Sie unter [Anwendungen mit Webpack bündeln](#)

Schritt 4: Beispielcode herunterladen

Verwenden Sie das Terminal, das Sie im vorherigen Schritt geöffnet haben, um Beispielcode für das SDK für JavaScript in die AWS Cloud9 Entwicklungsumgebung herunterzuladen. (Wenn das Terminal nicht bereits in der IDE geöffnet ist, öffnen Sie es, indem Sie `Window, New Terminal` (Fenster, Neues Terminal) auf der Menüleiste in der IDE auswählen.)

Führen Sie den folgenden Befehl aus, um den Beispielcode herunterzuladen. Mit diesem Befehl wird eine Kopie aller Codebeispiele, die in der offiziellen AWS SDK-Dokumentation verwendet werden, in das Stammverzeichnis Ihrer Umgebung heruntergeladen.

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

Codebeispiele für das -SDK für finden Sie im Fenster Umgebung JavaScript, um die zu öffnen `ENVIRONMENT_NAME\aws-doc-sdk-examples\javascriptv3\example_code/src`, wobei `ENVIRONMENT_NAME` der Name Ihrer AWS Cloud9 Entwicklungsumgebung ist.

Informationen zur Arbeit mit diesen und anderen Codebeispielen finden Sie unter [SDK für JavaScript Codebeispiele](#).

Schritt 5: Beispielcode ausführen und debuggen

Informationen zum Ausführen von Code in Ihrer AWS Cloud9 Entwicklungsumgebung finden Sie unter [Ausführen Ihres Codes](#) im AWS Cloud9 -Benutzerhandbuch.

Informationen zum Debuggen von Node.js-Code finden Sie unter [Debuggen Ihres Codes](#) im AWS Cloud9 -Benutzerhandbuch.

SDK für JavaScript (v3) -Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie AWS SDK for JavaScript (v3) mit verwenden AWS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Serviceübergreifende Beispiele sind Beispielanwendungen, die über mehrere AWS-Services hinweg arbeiten.

Beispiele

- [Aktionen und Szenarien mit dem SDK für JavaScript \(v3\)](#)
- [Serviceübergreifende Beispiele mit SDK für JavaScript \(v3\)](#)

Aktionen und Szenarien mit dem SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen, wie Aktionen ausgeführt und allgemeine Szenarien mithilfe von AWS SDK for JavaScript (v3) mit implementiert AWS-Services werden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Services

- [Auto Scaling Scaling-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Bedrock-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Bedrock Runtime-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Agenten für Amazon Bedrock, die SDK für JavaScript \(v3\) verwenden](#)

- [Beispiele für Agenten für Amazon Bedrock Runtime, die SDK für JavaScript \(v3\) verwenden](#)
- [CloudWatch Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [CloudWatch Beispiele für Ereignisse mit SDK für JavaScript \(v3\)](#)
- [CloudWatch Protokolliert Beispiele mit SDK für JavaScript \(v3\)](#)
- [CodeBuild Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Cognito Identity Provider mit SDK für JavaScript \(v3\)](#)
- [Amazon DocumentDB DocumentDB-Beispiele mit SDK für JavaScript \(v3\)](#)
- [DynamoDB-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon EC2 EC2-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Elastic Load Balancing — Beispiele für Version 2 mit SDK für JavaScript \(v3\)](#)
- [EventBridge Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [AWS Glue Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [HealthImaging Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [IAM-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Kinesis-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Lambda-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Personalize Personalize-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Personalize Events mit SDK für JavaScript \(v3\)](#)
- [Amazon Personalize Runtime-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Pinpoint Pinpoint-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Redshift Redshift-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon S3 S3-Beispiele mit SDK für JavaScript \(v3\)](#)
- [S3 Glacier-Beispiele mit SDK für JavaScript \(v3\)](#)
- [SageMaker Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Secrets Manager Manager-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon SES SES-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon SNS SNS-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon SQS SQS-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Step Functions Functions-Beispiele mit SDK für JavaScript \(v3\)](#)
- [AWS STS Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)

- [AWS Support Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Amazon Transcribe Transcribe-Beispiele mit SDK für JavaScript \(v3\)](#)

Auto Scaling Scaling-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Auto Scaling Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AttachLoadBalancerTargetGroups

Das folgende Codebeispiel zeigt, wie man es benutzt `AttachLoadBalancerTargetGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new AutoScalingClient({});
```

```
await client.send(  
  new AttachLoadBalancerTargetGroupsCommand({  
    AutoScalingGroupName: NAMES.autoScalingGroupName,  
    TargetGroupARNs: [state.targetGroupArn],  
  }),  
);
```

- Einzelheiten zur API finden Sie [AttachLoadBalancerTargetGroups](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};
```

```
// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
```

```

    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
    new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
    new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
        type: "confirm",
    }),
    new ScenarioAction(
        "handleConfirmDeployment",
        (c) => c.confirmDeployment === false && process.exit(),
    ),
    new ScenarioOutput(
        "creatingTable",
        MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioAction("createTable", async () => {
        const client = new DynamoDBClient({});
        await client.send(
            new CreateTableCommand({
                TableName: NAMES.tableName,
                ProvisionedThroughput: {

```

```
        ReadCapacityUnits: 5,
        WriteCapacityUnits: 5,
    },
    AttributeDefinitions: [
        {
            AttributeName: "MediaType",
            AttributeType: "S",
        },
        {
            AttributeName: "ItemId",
            AttributeType: "N",
        },
    ],
    KeySchema: [
        {
            AttributeName: "MediaType",
            KeyType: "HASH",
        },
        {
            AttributeName: "ItemId",
            KeyType: "RANGE",
        },
    ],
    }),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
        readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );
});
```

```
return client.send(
  new BatchWriteItemCommand({
    RequestItems: {
      [NAMES.tableName]: recommendations.map((item) => ({
        PutRequest: { Item: item },
      })),
    },
  }),
);
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );
  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
```

```
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  )),
);
state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  ),
});
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),

```

```
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
```

```
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
  );
}
```



```
    }),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
```

```
    */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
    ),
    new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
      type: "confirm",
    }),
    new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
    new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
    new ScenarioAction("getVpc", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
      const client = new EC2Client({});
      const { Vpcs } = await client.send(
        new DescribeVpcsCommand({
          Filters: [{ Name: "is-default", Values: ["true"] }],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
      state.defaultVpc = Vpcs[0].VpcId;
    }),
    new ScenarioOutput("gotVpc", (state) =>
      MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
    ),
    new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
    new ScenarioAction("getSubnets", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
      const client = new EC2Client({});
      const { Subnets } = await client.send(
        new DescribeSubnetsCommand({
          Filters: [
            { Name: "vpc-id", Values: [state.defaultVpc] },
            { Name: "availability-zone", Values: state.availabilityZoneNames },
            { Name: "default-for-az", Values: ["true"] },
          ],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
      state.subnets = Subnets.map((subnet) => subnet.SubnetId);
    }),
  ),
```

```
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
```

```
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    }),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
  const client = new ElasticLoadBalancingV2Client({});
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  ),
}),
```

```
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
const listener = Listeners[0];
state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
  "verifyInboundPort",
  /**
   *
   * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
   */
  async (state) => {
    const client = new EC2Client({});
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({
```

```

        Filters: [{ Name: "group-name", Values: ["default"] }],
    })),
);
if (!SecurityGroups) {
    state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
}
state.defaultSecurityGroup = SecurityGroups[0];

/**
 * @type {string}
 */
const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
state.myIp = ipResponse.trim();
const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
    ({ IpRanges }) =>
        IpRanges.some(
            ({ CidrIp }) =>
                CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
        ),
)
    .filter(({ IpProtocol }) => IpProtocol === "tcp")
    .filter(({ FromPort }) => FromPort === 80);

state.myIpRules = myIpRules;
},
),
new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return MESSAGES.foundIpRules.replace(
                "${IP_RULES}",
                JSON.stringify(state.myIpRules, null, 2),
            );
        } else {
            return MESSAGES.noIpRules;
        }
    },
),
new ScenarioInput(
    "shouldAddInboundRule",

```

```
/**
 * @param {{ myIpRules: any[] }} state
 */
(state) => {
  if (state.myIpRules.length > 0) {
    return false;
  } else {
    return MESSAGES.noIpRules;
  }
},
{ type: "confirm" },
),
new ScenarioAction(
  "addInboundRule",
  /**
   * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
   */
  async (state) => {
    if (!state.shouldAddInboundRule) {
      return;
    }

    const client = new EC2Client({});
    await client.send(
      new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.defaultSecurityGroup.GroupId,
        CidrIp: `${state.myIp}/32`,
        FromPort: 80,
        ToPort: 80,
        IpProtocol: "tcp",
      })),
    );
  },
),
new ScenarioOutput("addedInboundRule", (state) => {
  if (state.shouldAddInboundRule) {
    return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
  } else {
    return false;
  }
}),
new ScenarioOutput("verifyingEndpoint", (state) =>
  MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
```

```
),
new ScenarioAction("verifyEndpoint", async (state) => {
  try {
    const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
      axios.get(`http://${state.loadBalancerDns}`),
    );
    state.endpointResponse = JSON.stringify(response.data, null, 2);
  } catch (e) {
    state.verifyEndpointError = e;
  }
}),
new ScenarioOutput("verifiedEndpoint", (state) => {
  if (state.verifyEndpointError) {
    console.error(state.verifyEndpointError);
  } else {
    return MESSAGES.verifiedEndpoint.replace(
      "${ENDPOINT_RESPONSE}",
      state.endpointResponse,
    );
  }
}),
];
```

Erstellen Sie Schritte, um die Demo auszuführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
```



```
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    }
  }
);
```

```
    }
  } else {
    throw new Error(MESSAGES.demoFindLoadBalancerError);
  }
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-balancing-v2').TargetHealthDescription[] }} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
  }
);
```

```
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];
```

```
/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
```

```

    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioAction(
  "badCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
   */
  async (state) => {
    await createSsmOnlyInstanceProfile();
    const autoScalingClient = new AutoScalingClient({});
    const { AutoScalingGroups } = await autoScalingClient.send(

```

```
    new DescribeAutoScalingGroupsCommand({
      AutoScalingGroupNames: [NAMES.autoScalingGroupName],
    }),
  );
state.targetInstance = AutoScalingGroups[0].Instances[0];
// snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
// snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
// snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);
// snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );
};
```

```

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
   */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(

```

```

    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
},

```



```
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    })
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
```

```
    }),
  );
}),
new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
healthCheckLoop,
loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    ));
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
}
```

```

);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}

```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,

```

```
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
```

```
    console.error(state.deleteTableError);
    return MESSAGES.deleteTableError.replace(
      "${TABLE_NAME}",
      NAMES.tableName,
    );
  } else {
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }
}),
new ScenarioAction("deleteKeyPair", async (state) => {
  try {
    const client = new EC2Client({});
    await client.send(
      new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
    );
    unlinkSync(`${NAMES.keyPairName}.pem`);
  } catch (e) {
    state.deleteKeyPairError = e;
  }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
  if (state.deleteKeyPairError) {
    console.error(state.deleteKeyPairError);
    return MESSAGES.deleteKeyPairError.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  } else {
    return MESSAGES.deletedKeyPair.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
  try {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.detachPolicyFromRoleError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`
      );
    } else {
```

```
        await client.send(
            new DetachRolePolicyCommand({
                RoleName: NAMES.instanceRoleName,
                PolicyArn: policy.Arn,
            }),
        );
    }
} catch (e) {
    state.detachPolicyFromRoleError = e;
}
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
        console.error(state.detachPolicyFromRoleError);
        return MESSAGES.detachPolicyFromRoleError
            .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
        return MESSAGES.detachedPolicyFromRole
            .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
        state.deletePolicyError = new Error(
            `Policy ${NAMES.instancePolicyName} not found.`
        );
    } else {
        return client.send(
            new DeletePolicyCommand({
                PolicyArn: policy.Arn,
            }),
        );
    }
}),
new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
        console.error(state.deletePolicyError);
        return MESSAGES.deletePolicyError.replace(
            "${INSTANCE_POLICY_NAME}",
```

```
        NAMES.instancePolicyName,
    );
} else {
    return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
    );
}
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new RemoveRoleFromInstanceProfileCommand({
                RoleName: NAMES.instanceRoleName,
                InstanceProfileName: NAMES.instanceProfileName,
            }),
        );
    } catch (e) {
        state.removeRoleFromInstanceProfileError = e;
    }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
        console.error(state.removeRoleFromInstanceProfileError);
        return MESSAGES.removeRoleFromInstanceProfileError
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    } else {
        return MESSAGES.removedRoleFromInstanceProfile
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new DeleteRoleCommand({
                RoleName: NAMES.instanceRoleName,
            }),
        );
    } catch (e) {
        state.deleteInstanceRoleError = e;
    }
});
```

```
    }
  })),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  })),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  })),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    } else {
      return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
      );
    }
  })),
}
```



```
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  } else {
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
});
```

```
    );
  } else {
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
```

```
// snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
const client = new ElasticLoadBalancingV2Client({});
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
// snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  }
});
```

```
    } catch (e) {
      state.detachSsmOnlyRoleFromProfileError = e;
    }
  })),
  new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
      console.error(state.detachSsmOnlyRoleFromProfileError);
      return MESSAGES.detachSsmOnlyRoleFromProfileError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  })),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        })
      );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  })),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  })),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
```

```
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
      }),
    );
  } catch (e) {
    state.detachSsmOnlyAWSRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
  if (state.detachSsmOnlyAWSRolePolicyError) {
    console.error(state.detachSsmOnlyAWSRolePolicyError);
    return MESSAGES.detachSsmOnlyAWSRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  } else {
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
    console.error(state.deleteSsmOnlyInstanceProfileError);
    return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
```

```
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
    );
}
}),
new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
            new DeletePolicyCommand({
                PolicyArn: ssmOnlyPolicy.Arn,
            }),
        );
    } catch (e) {
        state.deleteSsmOnlyPolicyError = e;
    }
}),
new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
        console.error(state.deleteSsmOnlyPolicyError);
        return MESSAGES.deleteSsmOnlyPolicyError.replace(
            "${POLICY_NAME}",
            NAMES.ssmOnlyPolicyName,
        );
    } else {
        return MESSAGES.deletedSsmOnlyPolicy.replace(
            "${POLICY_NAME}",
            NAMES.ssmOnlyPolicyName,
        );
    }
}),
new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
        const iamClient = new IAMClient({});
        await iamClient.send(
            new DeleteRoleCommand({
                RoleName: NAMES.ssmOnlyRoleName,
            }),
        );
    } catch (e) {
        state.deleteSsmOnlyRoleError = e;
    }
}),
```

```
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
  if (state.deleteSsmOnlyRoleError) {
    console.error(state.deleteSsmOnlyRoleError);
    return MESSAGES.deleteSsmOnlyRoleError.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }
}),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    }
  }
}
```

```
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```


- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplaceIamInstanceProfileAssociation](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Amazon Bedrock-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Bedrock Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Bedrock beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });
```

```
export const main = async () => {
  const command = new ListFoundationModelsCommand({});

  const response = await client.send(command);
  const models = response.modelSummaries;

  console.log("Listing the available Bedrock foundation models:");

  for (let model of models) {
    console.log("=".repeat(42));
    console.log(` Model: ${model.modelId}`);
    console.log("-".repeat(42));
    console.log(` Name: ${model.modelName}`);
    console.log(` Provider: ${model.providerName}`);
    console.log(` Model ARN: ${model.modelArn}`);
    console.log(` Input modalities: ${model.inputModalities}`);
    console.log(` Output modalities: ${model.outputModalities}`);
    console.log(` Supported customizations: ${model.customizationsSupported}`);
    console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
    console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
    console.log("=".repeat(42) + "\n");
  }

  const active = models.filter(
    (m) => m.modelLifecycle.status === "ACTIVE",
  ).length;
  const legacy = models.filter(
    (m) => m.modelLifecycle.status === "LEGACY",
  ).length;

  console.log(
    `There are ${active} active and ${legacy} legacy foundation models in  

    ${REGION}.`,
  );

  return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- Einzelheiten zur API finden Sie [ListFoundationModels](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)

Aktionen

GetFoundationModel

Das folgende Codebeispiel zeigt die Verwendung `GetFoundationModel`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erfahren Sie mehr über ein Gründungsmodell.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 * models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();
```

```
const command = new GetFoundationModelCommand({
  modelIdentifier: "amazon.titan-embed-text-v1",
});

const response = await client.send(command);

return response.modelDetails;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- Einzelheiten zur API finden Sie [GetFoundationModel](#) unter AWS SDK for JavaScript API-Referenz.

ListFoundationModels

Das folgende Codebeispiel zeigt die Verwendung `ListFoundationModels`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listet die verfügbaren Fundamentmodelle auf.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";
```

```
/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
 models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
    // byProvider: 'STRING_VALUE',
    // byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
    // byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
    // byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
  };

  const command = new ListFoundationModelsCommand(input);

  const response = await client.send(command);

  return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const models = await listFoundationModels();
  console.log(models);
}
```

- Einzelheiten zur API finden Sie [ListFoundationModels](#) unter AWS SDK for JavaScript API-Referenz.

Amazon Bedrock Runtime-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Bedrock Runtime Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, über den Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Bedrock beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";
```

```
const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    }),
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;

  if (responses.length === 1) {
    console.log(`Response: ${responses[0].text}`);
  } else {
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }
}
```



```
console.log(`\nNumber of input tokens:  ${responseBody.usage.input_tokens}`);
console.log(`Number of output tokens:  ${responseBody.usage.output_tokens}`);
};

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await hello();
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [AI21 Labs Jurassic-2](#)
- [Amazon Titan Text](#)
- [Anthropic Claude](#)
- [Cohere Command](#)
- [Meta-Lama](#)
- [Mistral KI](#)
- [Szenarien](#)

AI21 Labs Jurassic-2

Converse

Das folgende Codebeispiel zeigt, wie mithilfe der Converse-API von Bedrock eine Textnachricht an AI21 Labs Jurassic-2 gesendet wird.

JavaScript SDK für (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an AI21 Labs Jurassic-2.

```
// Use the Conversation API to send a text message to AI21 Labs Jurassic-2.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Jurassic-2 Mid.
const modelId = "ai21.j2-mid-v1";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- [Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK for JavaScript](#)

InvokeModel

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an AI21 Labs Jurassic-2 gesendet wird.

SDK für (v3) JavaScript

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Data
 * @property {string} text
 *
 * @typedef {Object} Completion
 * @property {Data} data
 *
 * @typedef {Object} ResponseBody
 * @property {Completion[]} completions
 */

/**
 * Invokes an AI21 Labs Jurassic-2 model.
 */
```

```
* @param {string} prompt - The input text prompt for the model to complete.
* @param {string} [modelId] - The ID of the model to use. Defaults to "ai21.j2-mid-
v1".
*/
export const invokeModel = async (prompt, modelId = "ai21.j2-mid-v1") => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    prompt,
    maxTokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s).
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.completions[0].data.text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time...";
  const modelId = FoundationModels.JURASSIC2_MID.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

```
}  
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

Amazon Titan Text

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Titan Text senden.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Titan Text.

```
// Use the Conversation API to send a text message to Amazon Titan Text.  
  
import {  
  BedrockRuntimeClient,  
  ConverseCommand,  
} from "@aws-sdk/client-bedrock-runtime";  
  
// Create a Bedrock Runtime client in the AWS Region you want to use.  
const client = new BedrockRuntimeClient({ region: "us-east-1" });  
  
// Set the model ID, e.g., Titan Text Premier.  
const modelId = "amazon.titan-text-premier-v1:0";  
  
// Start a conversation with the user message.  
const userMessage =  
  "Describe the purpose of a 'hello world' program in one line.";  
const conversation = [  
  {  
    role: "user",
```

```
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie unter [Converse](#) in AWS SDK for JavaScript der API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Titan Text senden und den Antwortstream in Echtzeit verarbeiten.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Titan Text und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Amazon Titan Text.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Titan Text Premier.
const modelId = "amazon.titan-text-premier-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

```
}  
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK for JavaScript API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Amazon Titan Text senden.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
  
import { fileURLToPath } from "url";  
  
import { FoundationModels } from "../../config/foundation_models.js";  
import {  
  BedrockRuntimeClient,  
  InvokeModelCommand,  
} from "@aws-sdk/client-bedrock-runtime";  
  
/**  
 * @typedef {Object} ResponseBody  
 * @property {Object[]} results  
 */  
  
/**  
 * Invokes an Amazon Titan Text generation model.  
 *  
 * @param {string} prompt - The input text prompt for the model to complete. */
```



```
* @param {string} [modelId] - The ID of the model to use. Defaults to
"amazon.titan-text-express-v1".
*/
export const invokeModel = async (
  prompt,
  modelId = "amazon.titan-text-express-v1",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    inputText: prompt,
    textGenerationConfig: {
      maxTokenCount: 4096,
      stopSequences: [],
      temperature: 0,
      topP: 1,
    },
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.results[0].outputText;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time...";
  const modelId = FoundationModels.TITAN_TEXT_G1_EXPRESS.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);
}
```

```
try {
  console.log("-".repeat(53));
  const response = await invokeModel(prompt, modelId);
  console.log(response);
} catch (err) {
  console.log(err);
}
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

Anthropic Claude

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";
```

```
// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz.AWS SDK for JavaScript

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden und den Antwortstream in Echtzeit verarbeiten.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```

```
// Send the command to the model and wait for the response
const response = await client.send(command);

// Extract and print the streamed response text in real-time.
for await (const item of response.stream) {
  if (item.contentBlockDelta) {
    process.stdout.write(item.contentBlockDelta.delta?.text);
  }
}
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in AWS SDK for JavaScript der API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an Anthropic Claude gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
}
```

```
    InvokeModelWithResponseStreamCommand,
  } from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
  };
};
```

```
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
  contentType: "application/json",
  body: JSON.stringify(payload),
  modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {MessagesResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
  };
};
```

```
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

// Invoke Claude with the payload and wait for the API to respond.
const command = new InvokeModelWithResponseStreamCommand({
  contentType: "application/json",
  body: JSON.stringify(payload),
  modelId,
});
const apiResponse = await client.send(command);

let completeMessage = "";

// Decode and process the response stream
for await (const item of apiResponse.body) {
  /** @type Chunk */
  const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
  const chunk_type = chunk.type;

  if (chunk_type === "content_block_delta") {
    const text = chunk.delta.text;
    completeMessage = completeMessage + text;
    process.stdout.write(text);
  }
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time...";
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
```



```
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

InvokeModelWithResponseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Modelle von Anthropic Claude senden und den Antwortstream drucken.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
```

```
* @typedef {Object} ResponseContent
* @property {string} text
*
* @typedef {Object} MessagesResponseBody
* @property {ResponseContent[]} content
*
* @typedef {Object} Delta
* @property {string} text
*
* @typedef {Object} Message
* @property {string} role
*
* @typedef {Object} Chunk
* @property {string} type
* @property {Delta} delta
* @property {Message} message
*/

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      }
    ]
  }
}
```

```
    },
  ],
};

// Invoke Claude with the payload and wait for the response.
const command = new InvokeModelCommand({
  contentType: "application/json",
  body: JSON.stringify(payload),
  modelId,
});
const apiResponse = await client.send(command);

// Decode and return the response(s)
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {MessagesResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.content[0].text;
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
  claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
  "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
```

```
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;

    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time...";
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log("\n" + "-".repeat(53));
    console.log("Final structured response:");
  }
}
```

```
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Einzelheiten zur API finden Sie [InvokeModelWithResponseStream](#) unter AWS SDK for JavaScript API-Referenz.

Cohere Command

Converse

Das folgende Codebeispiel zeigt, wie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
```

```
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);


  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz.AWS SDK for JavaScript

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command senden und den Antwortstream in Echtzeit verarbeiten.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```

```
// Send the command to the model and wait for the response
const response = await client.send(command);

// Extract and print the streamed response text in real-time.
for await (const item of response.stream) {
  if (item.contentBlockDelta) {
    process.stdout.write(item.contentBlockDelta.delta?.text);
  }
}
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in AWS SDK for JavaScript der API-Referenz.

Meta-Lama

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama senden.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";
```



```
// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz.AWS SDK for JavaScript

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama senden und den Antwortstream in Echtzeit verarbeiten.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```

```
// Send the command to the model and wait for the response
const response = await client.send(command);

// Extract and print the streamed response text in real-time.
for await (const item of response.stream) {
  if (item.contentBlockDelta) {
    process.stdout.write(item.contentBlockDelta.delta?.text);
  }
}
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in AWS SDK for JavaScript der API-Referenz.

InvokeModel: Lama 2

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an Meta Llama 2 gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Send a prompt to Meta Llama 2 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
```

```
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 2 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-2
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

InvokeModel: Lama 3

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an Meta Llama 3 gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
```

```
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);


// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

InvokeModelWithResponseStream: Lama 2

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Meta Llama 2 senden und den Antwortstream drucken.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
// Send a prompt to Meta Llama 2 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 2 Chat 13B.
const modelId = "meta.llama2-13b-chat-v1";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 2's prompt format.
const prompt = `[INST] ${userMessage} [/INST]`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
```

```
new InvokeModelWithResponseStreamCommand({
  contentType: "application/json",
  body: JSON.stringify(request),
  modelId,
}),
);

// Extract and print the response stream in real-time.
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Einzelheiten zur API finden Sie [InvokeModelWithResponseStream](#) unter AWS SDK for JavaScript API-Referenz.

InvokeModelWithResponseStream: Llama 3

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Meta Llama 3 senden und den Antwortstream drucken.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.
```



```
import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 8B Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Extract and print the response stream in real-time.
```

```
for await (const event of responseStream.body) {
  /** @type {{ generation: string }} */
  const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  if (chunk.generation) {
    process.stdout.write(chunk.generation);
  }
}

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Einzelheiten zur API finden Sie [InvokeModelWithResponseStream](#) unter AWS SDK for JavaScript API-Referenz.

Mistral KI

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral senden.

SDK für (v3) JavaScript

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
```

```
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie unter [Converse](#) in der API-Referenz.AWS SDK for JavaScript

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral senden und den Antwortstream in Echtzeit verarbeiten.

SDK für (v3) JavaScript

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
```

```
// Send the command to the model and wait for the response
const response = await client.send(command);

// Extract and print the streamed response text in real-time.
for await (const item of response.stream) {
  if (item.contentBlockDelta) {
    process.stdout.write(item.contentBlockDelta.delta?.text);
  }
}
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie in der API-Referenz. [ConverseStream](#) AWS SDK for JavaScript

InvokeModel

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an Mistral-Modelle gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
}
```

```
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response.

```

```
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time...";
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) unter AWS SDK for JavaScript API-Referenz.

Szenarien

Rufen Sie mehrere Foundation-Modelle auf Amazon Bedrock auf

Das folgende Codebeispiel zeigt, wie Sie eine Aufforderung vorbereiten und an eine Vielzahl von Large-Language Models (LLMs) auf Amazon Bedrock senden

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { FoundationModels } from "../config/foundation_models.js";

/**
 * @typedef {Object} ModelConfig
 * @property {Function} module
 * @property {Function} invoker
 * @property {string} modelId
 * @property {string} modelName
 */

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */

```



```
(c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
{ slow: false },
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();
    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);

const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c
   */
  (c) => c.response,
  { slow: false },
);

const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
  greeting,
  selectModel,
  enterPrompt,
  printDetails,
  invokeModel,
  printResponse,
]);

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  scenario.run();
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [InvokeModel](#)
 - [InvokeModelWithResponseStream](#)

Beispiele für Agenten für Amazon Bedrock, die SDK für JavaScript (v3) verwenden

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Agents for Amazon Bedrock verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen finden, wie Sie den Code im Kontext einrichten und ausführen können.

Erste Schritte

Hallo Agenten für Amazon Bedrock

Das folgende Codebeispiel zeigt, wie Sie mit Agents for Amazon Bedrock beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
```

```
* @typedef {Object} AgentSummary
*/

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 * Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 * region.
 * It then retrieves a list of existing agents using the streamlined paginator
 * approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
 * - Use of the Bedrock Agents client to initialize and communicate with the AWS
 * service.
 * - Listing resources in a paginated response pattern.
 * - Accessing an individual resource using a command object.
 *
 * @returns {Promise<void>} A promise that resolves when the function has completed
 * execution.
 */
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log(`Retrieving the list of existing agents...`);
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
```

```
const agentId = agentSummary.agentId;
console.log("=".repeat(68));
console.log(`Retrieving agent with ID: ${agentId}:`);
console.log("-".repeat(68));

const command = new GetAgentCommand({ agentId });
const response = await client.send(command);
const agent = response.agent;

console.log(` Name: ${agent.agentName}`);
console.log(` Status: ${agent.agentStatus}`);
console.log(` ARN: ${agent.agentArn}`);
console.log(` Foundation model: ${agent.foundationModel}`);
}
}
console.log("=".repeat(68));
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [GetAgent](#)
 - [ListAgents](#)

Themen

- [Aktionen](#)

Aktionen

CreateAgent

Das folgende Codebeispiel zeigt, wie man es benutztCreateAgent.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie einen -Agenten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  CreateAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Creates an Amazon Bedrock Agent.
 *
 * @param {string} agentName - A name for the agent that you create.
 * @param {string} foundationModel - The foundation model to be used by the agent
you create.
 * @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
required by the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
containing details of the created agent.
 */
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
```

```
        foundationModel,
        agentResourceRoleArn,
    });
    const response = await client.send(command);

    return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    // Replace the placeholders for agentName and accountId, and roleName with a
    // unique name for the new agent,
    // the id of your AWS account, and the name of an existing execution role that the
    // agent can use inside your account.
    // For foundationModel, specify the desired model. Ensure to remove the brackets
    // '[]' before adding your data.

    // A string (max 100 chars) that can include letters, numbers, dashes '-', and
    // underscores '_'.
    const agentName = "[your-bedrock-agent-name]";

    // Your AWS account id.
    const accountId = "[123456789012]";

    // The name of the agent's execution role. It must be prefixed by
    // `AmazonBedrockExecutionRoleForAgents_`.
    const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

    // The ARN for the agent's execution role.
    // Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
    const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

    // Specify the model for the agent. Change if a different model is preferred.
    const foundationModel = "anthropic.claude-v2";

    // Check for unresolved placeholders in agentName and roleArn.
    checkForPlaceholders([agentName, roleArn]);

    console.log(`Creating a new agent...`);

    const agent = await createAgent(agentName, foundationModel, roleArn);
    console.log(agent);
}
```

- Einzelheiten zur API finden Sie [CreateAgent](#) in der AWS SDK for JavaScript API-Referenz.

DeleteAgent

Das folgende Codebeispiel zeigt die Verwendung `DeleteAgent`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Lösche einen Agenten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
  and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
```

```
    return client.send(command);
  };

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}
```

- Einzelheiten zur API finden Sie [DeleteAgent](#) unter AWS SDK for JavaScript API-Referenz.

GetAgent

Das folgende Codebeispiel zeigt die Verwendung `GetAgent`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Besorgen Sie sich einen Agenten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
```



```
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
  containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Retrieving agent with ID ${agentId}...`);

  const agent = await getAgent(agentId);
  console.log(agent);
}
```

- Einzelheiten zur API finden Sie [GetAgent](#) unter AWS SDK for JavaScript API-Referenz.

ListAgentActionGroups

Das folgende Codebeispiel zeigt die Verwendung `ListAgentActionGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listet die Aktionsgruppen für einen Agenten auf.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
```

```
const client = new BedrockAgentClient({ region });

// Create a paginator configuration
const paginatorConfig = {
  client,
  pageSize: 10, // optional, added for demonstration purposes
};

const params = { agentId, agentVersion };

const pages = paginateListAgentActionGroups(paginatorConfig, params);

// Paginate until there are no more results
const actionGroupSummaries = [];
for await (const page of pages) {
  actionGroupSummaries.push(...page.actionGroupSummaries);
}

return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 * ListAgentActionGroupsCommand.
 *
 * * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 * * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * * @param {string} agentId - The unique identifier of the agent.
 * * @param {string} agentVersion - The version of the agent.
 * * @param {string} [region='us-east-1'] - The AWS region in use.
 * * @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
 */
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
```

```
const actionGroupSummaries = [];
do {
  const command = new ListAgentActionGroupsCommand({
    agentId,
    agentVersion,
    nextToken,
    maxResults: 10, // optional, added for demonstration purposes
  });

  /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
  const response = await client.send(command);

  for (const actionGroup of response.actionGroupSummaries || []) {
    actionGroupSummaries.push(actionGroup);
  }

  nextToken = response.nextToken;
} while (nextToken);

return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
  // id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
  // 'DRAFT').
  const agentVersion = "[DRAFT]";

  // Check for unresolved placeholders in agentId and agentVersion.
  checkForPlaceholders([agentId, agentVersion]);

  console.log("=".repeat(68));
  console.log(
    "Listing agent action groups using ListAgentActionGroupsCommand:",
  );

  for (const actionGroup of await listAgentActionGroupsWithCommandObject(
```

```
    agentId,  
    agentVersion,  
  )) {  
    console.log(actionGroup);  
  }  
  
  console.log("=".repeat(68));  
  console.log(  
    "Listing agent action groups using the paginateListAgents function:",  
  );  
  for (const actionGroup of await listAgentActionGroupsWithPaginator(  
    agentId,  
    agentVersion,  
  )) {  
    console.log(actionGroup);  
  }  
}
```

- Einzelheiten zur API finden Sie [ListAgentActionGroups](#) unter AWS SDK for JavaScript API-Referenz.

ListAgents

Das folgende Codebeispiel zeigt die Verwendung `ListAgents`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listet die Agenten auf, die zu einem Konto gehören.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
  
import { fileURLToPath } from "url";  
  
import {
```

```
    BedrockAgentClient,
    ListAgentsCommand,
    paginateListAgents,
  } from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 * function.
 *
 * * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 * ListAgentsCommand.
 *
 * * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 * * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 */
```

```
*
* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<AgentSummary[]>} An array of agent summaries.
*/
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
  console.log("Listing agents using ListAgentsCommand:");
  for (const agent of await listAgentsWithCommandObject()) {
    console.log(agent);
  }

  console.log("=".repeat(68));
  console.log("Listing agents using the paginateListAgents function:");
  for (const agent of await listAgentsWithPaginator()) {
    console.log(agent);
  }
}
```

- Einzelheiten zur API finden Sie [ListAgents](#) unter AWS SDK for JavaScript API-Referenz.

Beispiele für Agenten für Amazon Bedrock Runtime, die SDK für JavaScript (v3) verwenden

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Agents for Amazon Bedrock Runtime verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden können.

Themen

- [Aktionen](#)

Aktionen

InvokeAgent

Das folgende Codebeispiel zeigt, wie man es benutzt InvokeAgent.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
  
import {  
  BedrockAgentRuntimeClient,  
  InvokeAgentCommand,  
}
```



```
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (let chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
    }
  }
}
```

```
    console.log(chunk);
    const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
    completion += decodedResponse;
  }

  return { sessionId: sessionId, completion };
} catch (err) {
  console.error(err);
}
};

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Einzelheiten zur API finden Sie [InvokeAgent](#) in der AWS SDK for JavaScript API-Referenz.

CloudWatch Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden CloudWatch.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

DeleteAlarms

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteAlarms`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```


Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAlarms](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};


cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAlarms](#) in der AWS SDK for JavaScript API-Referenz.

DescribeAlarmsForMetric

Das folgende Codebeispiel zeigt die Verwendung `DescribeAlarmsForMetric`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeAlarmsForMetric](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });


cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeAlarmsForMetric](#) in der AWS SDK for JavaScript API-Referenz.

DisableAlarmActions

Das folgende Codebeispiel zeigt die Verwendung `DisableAlarmActions`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DisableAlarmActions](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DisableAlarmActions](#) in der AWS SDK for JavaScript API-Referenz.

EnableAlarmActions

Das folgende Codebeispiel zeigt die Verwendung `EnableAlarmActions`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```


Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [EnableAlarmActions](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
console.log("Alarm action added", data);
var paramsEnableAlarmAction = {
  AlarmNames: [params.AlarmName],
};
cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action enabled", data);
  }
});
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [EnableAlarmActions](#) in der AWS SDK for JavaScript API-Referenz.

ListMetrics

Das folgende Codebeispiel zeigt die Verwendung `ListMetrics`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
  // Use the AWS console to see available namespaces and metric names. Custom
  // metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
  // viewing_metrics_with_cloudwatch.html
```

```
const command = new ListMetricsCommand({
  Dimensions: [
    {
      Name: "LogGroupName",
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
});

return client.send(command);
};
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListMetrics](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
```

```
Dimensions: [  
  {  
    Name: "LogGroupName" /* required */,  
  },  
],  
MetricName: "IncomingLogEvents",  
Namespace: "AWS/Logs",  
};  
  
cw.listMetrics(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Metrics", JSON.stringify(data.Metrics));  
  }  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListMetrics](#) in der AWS SDK for JavaScript API-Referenz.

PutMetricAlarm

Das folgende Codebeispiel zeigt die Verwendung `PutMetricAlarm`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";  
import { client } from "../libs/client.js";  
  
const run = async () => {  
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.  
  const command = new PutMetricAlarmCommand({
```

```
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
        the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```


Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricAlarm](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

```
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricAlarm](#) in der AWS SDK for JavaScript API-Referenz.

PutMetricData

Das folgende Codebeispiel zeigt die Verwendung `PutMetricData`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
  });
```



```
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricData](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
```

```
MetricData: [
  {
    MetricName: "PAGES_VISITED",
    Dimensions: [
      {
        Name: "UNIQUE_PAGES",
        Value: "URLS",
      },
    ],
    Unit: "None",
    Value: 1.0,
  },
],
Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricData](#) in der AWS SDK for JavaScript API-Referenz.

CloudWatch Beispiele für Ereignisse mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit CloudWatch Ereignissen verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt, wie man es benutztPutEvents.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
},
```

```
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK for JavaScript API-Referenz.

PutRule

Das folgende Codebeispiel zeigt die Verwendung `PutRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  // Request parameters for PutRule.
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/API_PutRule.html#API_PutRule_RequestParameters
  const command = new PutRuleCommand({
    Name: process.env.CLOUDWATCH_EVENTS_RULE,

    // The event pattern for the rule.
    // Example: {"source": ["my.app"]}
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,
```

```
// The state of the rule. Valid values: ENABLED, DISABLED
State: "ENABLED",
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
```

```
Name: "DEMO_EVENT",
RoleArn: "IAM_ROLE_ARN",
ScheduleExpression: "rate(5 minutes)",
State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK for JavaScript API-Referenz.

PutTargets

Das folgende Codebeispiel zeigt die Verwendung `PutTargets`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
```

```
    {
      Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
      // The ID of the target. Choose a unique ID for each target.
      Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
    },
  ],
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
```



```
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK for JavaScript API-Referenz.

CloudWatch Protokolliert Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit CloudWatch Logs verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)


- [Szenarien](#)

Aktionen

CreateLogGroup

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateLogGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [CreateLogGroup](#) in der AWS SDK for JavaScript API-Referenz.

DeleteLogGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteLogGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};


export default run();
```

- Einzelheiten zur API finden Sie [DeleteLogGroup](#) in der AWS SDK for JavaScript API-Referenz.

DeleteSubscriptionFilter

Das folgende Codebeispiel zeigt die Verwendung `DeleteSubscriptionFilter`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [DeleteSubscriptionFilter](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
```

```
    logGroupName: "LOG_GROUP",
  };

  cwl.deleteSubscriptionFilter(params, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  });
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteSubscriptionFilter](#) in der AWS SDK for JavaScript API-Referenz.

DescribeLogGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeLogGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups && page.logGroups.every((lg) => !!lg)) {
```

```
    logGroups.push(...page.logGroups);
  }
}

console.log(logGroups);
return logGroups;
};
```

- Einzelheiten zur API finden Sie [DescribeLogGroups](#) in der AWS SDK for JavaScript API-Referenz.

DescribeSubscriptionFilters

Das folgende Codebeispiel zeigt die Verwendung `DescribeSubscriptionFilters`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
}
```

```
};  
  
export default run();
```

- Einzelheiten zur API finden Sie [DescribeSubscriptionFilters](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create the CloudWatchLogs service object  
var cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });  
  
var params = {  
  logGroupName: "GROUP_NAME",  
  limit: 5,  
};  
  
cw1.describeSubscriptionFilters(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data.subscriptionFilters);  
  }  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeSubscriptionFilters](#) in der AWS SDK for JavaScript API-Referenz.

GetQueryResults

Das folgende Codebeispiel zeigt die Verwendung `GetQueryResults`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Einzelheiten zur API finden Sie [GetQueryResults](#) in der AWS SDK for JavaScript API-Referenz.

PutSubscriptionFilter

Das folgende Codebeispiel zeigt die Verwendung `PutSubscriptionFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
```



```
// An ARN of a same-account Kinesis stream, Kinesis Firehose
// delivery stream, or Lambda function.
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
SubscriptionFilters.html
destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

// A name for the filter.
filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

// A filter pattern for subscribing to a filtered stream of log events.
// https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
FilterAndPatternSyntax.html
filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

// The name of the log group. Messages in this group matching the filter pattern
// will be sent to the destination ARN.
logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

- Einzelheiten zur API finden Sie [PutSubscriptionFilter](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutSubscriptionFilter](#) in der AWS SDK for JavaScript API-Referenz.

StartLiveTail

Das folgende Codebeispiel zeigt die Verwendung `StartLiveTail`.

SDK für JavaScript (v3)

Binden Sie die erforderlichen Dateien ein.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Behandelt die Ereignisse aus der Live Tail-Sitzung.

```
async function handleResponseAsync(response) {
  try {
```

```
for await (const event of response.responseStream) {
  if (event.sessionStart !== undefined) {
    console.log(event.sessionStart);
  } else if (event.sessionUpdate !== undefined) {
    for (const logEvent of event.sessionUpdate.sessionResults) {
      const timestamp = logEvent.timestamp;
      const date = new Date(timestamp);
      console.log("[ " + date + " ] " + logEvent.message);
    }
  } else {
    console.error("Unknown event type");
  }
}
} catch (err) {
  // On-stream exceptions are captured here
  console.error(err)
}
}
```

Starten Sie die Live-Tail-Sitzung.

```
const client = new CloudWatchLogsClient();

const command = new StartLiveTailCommand({
  logGroupIdentifiers: logGroupIdentifiers,
  logStreamNames: logStreamNames,
  logEventFilterPattern: filterPattern
});
try{
  const response = await client.send(command);
  handleResponseAsync(response);
} catch (err){
  // Pre-stream exceptions are captured here
  console.log(err);
}
```

Beenden Sie die Live-Tail-Sitzung nach Ablauf einer gewissen Zeit.

```
/* Set a timeout to close the client. This will stop the Live Tail session. */
setTimeout(function() {
  console.log("Client timeout");
}, 10000);
```

```
    client.destroy();
  }, 10000);
```

- Einzelheiten zur API finden Sie [StartLiveTail](#) in der AWS SDK for JavaScript API-Referenz.

StartQuery

Das folgende Codebeispiel zeigt die Verwendung `StartQuery`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
    }
  }
}
```

```
        throw new DateOutOfBoundsError(message);
    }

    throw err;
}
}
```

- Einzelheiten zur API finden Sie [StartQuery](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Führen Sie eine umfangreiche Abfrage aus

Das folgende Codebeispiel zeigt, wie CloudWatch Logs verwendet werden kann, um mehr als 10.000 Datensätze abzufragen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Das ist der Einstiegspunkt.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
    throw new Error(
        "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
    );
}

const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
```

```

    logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
    dateRange: [
      new Date(parseInt(process.env.QUERY_START_DATE)),
      new Date(parseInt(process.env.QUERY_END_DATE)),
    ],
  });

  await cloudWatchQuery.run();

  console.log(
    `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found:
    ${cloudWatchQuery.results.length}`
  );

```

Dies ist eine Klasse, die Abfragen bei Bedarf in mehrere Schritte aufteilt.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
   * { limit: number } }} config
   */
  constructor(client, { logGroupNames, dateRange, queryConfig }) {
    this.client = client;
    /**
     * All log groups are queried.

```

```
    */
    this.logGroupNames = logGroupNames;

    /**
     * The inclusive date range that is queried.
     */
    this.dateRange = dateRange;

    /**
     * CloudWatch Logs never returns more than 10,000 logs.
     */
    this.limit = queryConfig?.limit ?? 10000;

    /**
     * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
     */
    this.results = [];
  }

  /**
   * Run the query.
   */
  async run() {
    this.secondsElapsed = 0;
    const start = new Date();
    this.results = await this._largeQuery(this.dateRange);
    const end = new Date();
    this.secondsElapsed = (end - start) / 1000;
    return this.results;
  }

  /**
   * Recursively query for logs.
   * @param {[Date, Date]} dateRange
   * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
   */
  async _largeQuery(dateRange) {
    const logs = await this._query(dateRange, this.limit);

    console.log(
      `Query date range: ${dateRange
        .map((d) => d.toISOString())
        .join(" to ")}. Found ${logs.length} logs.`
    );
  }
};
```

```
    if (logs.length < this.limit) {
      return logs;
    }

    const lastLogDate = this._getLastLogDate(logs);
    const offsetLastLogDate = new Date(lastLogDate);
    offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
    const subDateRange = [offsetLastLogDate, dateRange[1]];
    const [r1, r2] = splitDateRange(subDateRange);
    const results = await Promise.all([
      this._largeQuery(r1),
      this._largeQuery(r2),
    ]);
    return [logs, ...results].flat();
  }

  /**
   * Find the most recent log in a list of logs.
   * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
   */
  _getLastLogDate(logs) {
    const timestamps = logs
      .map(
        (log) =>
          log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
      )
      .filter((t) => !!t)
      .map((t) => `${t}Z`)
      .sort();

    if (!timestamps.length) {
      throw new Error("No timestamp found in logs.");
    }

    return new Date(timestamps[timestamps.length - 1]);
  }

  // snippet-start:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]
  /**
   * Simple wrapper for the GetQueryResultsCommand.
   * @param {string} queryId
   */
  _getQueryResults(queryId) {
```



```
    return this.client.send(new GetQueryResultsCommand({ queryId }));
  }
// snippet-end:[javascript.v3.cloudwatch-logs.actions.GetQueryResults]

/**
 * Starts a query and waits for it to complete.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 */
async _query(dateRange, maxLogs) {
  try {
    const { queryId } = await this._startQuery(dateRange, maxLogs);
    const { results } = await this._waitUntilQueryDone(queryId);
    return results ?? [];
  } catch (err) {
    /**
     * This error is thrown when StartQuery returns an error indicating
     * that the query's start or end date occur before the log group was
     * created.
     */
    if (err instanceof DateOutOfBoundsError) {
      return [];
    } else {
      throw err;
    }
  }
}

// snippet-start:[javascript.v3.cloudwatch-logs.actions.StartQuery]
/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
      })
    );
  }
}
```

```
        limit: maxLogs,
      )),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}
// snippet-end:[javascript.v3.cloudwatch-logs.actions.StartQuery]

/**
 * Call GetQueryResultsCommand until the query is done.
 * @param {string} queryId
 */
_waitUntilQueryDone(queryId) {
  const getResults = async () => {
    const results = await this._getQueryResults(queryId);
    const queryDone = [
      "Complete",
      "Failed",
      "Cancelled",
      "Timeout",
      "Unknown",
    ].includes(results.status);

    return { queryDone, results };
  };

  return retry(
    { intervalInMs: 1000, maxRetries: 60, quiet: true },
    async () => {
      const { queryDone, results } = await getResults();
      if (!queryDone) {
        throw new Error("Query not done.");
      }
    }

    return results;
  );
}
```

```
    },  
  );  
}  
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [GetQueryResults](#)
 - [StartQuery](#)

CodeBuild Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden CodeBuild.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateProject

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateProject`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Ein Projekt zu erstellen.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
      // and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
```

```
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
    },
    name: projectName,
    // A role ARN with permission to create a CodeBuild project, write to the
artifact location, and write CloudWatch logs.
    serviceRole: roleArn,
    source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
    },
}),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   project: {
//     arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
//     artifacts: {
//       encryptionDisabled: false,
//       location: 'xxxxxx-xxxxxxx-xxxxxx',
//       name: 'MyCodeBuilder',
//       namespaceType: 'NONE',
//       packaging: 'NONE',
//       type: 'S3'
//     },
//     badge: { badgeEnabled: false },
//     cache: { type: 'NO_CACHE' },
//     created: 2023-08-18T14:46:48.979Z,
//     encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//     environment: {
```

```
//      computeType: 'BUILD_GENERAL1_SMALL',
//      environmentVariables: [],
//      image: 'aws/codebuild/standard:7.0',
//      imagePullCredentialsType: 'CODEBUILD',
//      privilegedMode: false,
//      type: 'LINUX_CONTAINER'
//    },
//    lastModified: 2023-08-18T14:46:48.979Z,
//    name: 'MyCodeBuilder',
//    projectVisibility: 'PRIVATE',
//    queuedTimeoutInMinutes: 480,
//    serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
//    source: {
//      insecureSsl: false,
//      location: 'https://...',
//      reportBuildStatus: false,
//      type: 'GITHUB'
//    },
//    timeoutInMinutes: 60
//  }
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateProject](#) in der AWS SDK for JavaScript API-Referenz.

Beispiele für Amazon Cognito Identity Provider mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Amazon Cognito Identity Provider verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon Cognito

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon Cognito.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- Einzelheiten zur API finden Sie [ListUserPools](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AdminGetUser

Das folgende Codebeispiel zeigt die Verwendung `AdminGetUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });


  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AdminGetUser](#) in der AWS SDK for JavaScript API-Referenz.

AdminInitiateAuth

Das folgende Codebeispiel zeigt die Verwendung `AdminInitiateAuth`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });


  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AdminInitiateAuth](#) in der AWS SDK for JavaScript API-Referenz.

AdminRespondToAuthChallenge

Das folgende Codebeispiel zeigt die Verwendung `AdminRespondToAuthChallenge`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
```

```
username,
totp,
session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AdminRespondToAuthChallenge](#) in der AWS SDK for JavaScript API-Referenz.

AssociateSoftwareToken

Das folgende Codebeispiel zeigt die Verwendung `AssociateSoftwareToken`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });
};
```

```
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [AssociateSoftwareToken](#) in der AWS SDK for JavaScript API-Referenz.

ConfirmDevice

Das folgende Codebeispiel zeigt die Verwendung `ConfirmDevice`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {  
    const client = new CognitoIdentityProviderClient({});  
  
    const command = new ConfirmDeviceCommand({  
        DeviceKey: deviceKey,  
        AccessToken: accessToken,  
        DeviceSecretVerifierConfig: {  
            PasswordVerifier: passwordVerifier,  
            Salt: salt,  
        },  
    });  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [ConfirmDevice](#) in der AWS SDK for JavaScript API-Referenz.

ConfirmSignUp

Das folgende Codebeispiel zeigt die Verwendung `ConfirmSignUp`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });


  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ConfirmSignUp](#) in der AWS SDK for JavaScript API-Referenz.

InitiateAuth

Das folgende Codebeispiel zeigt die Verwendung `InitiateAuth`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
```

```
AuthParameters: {
  USERNAME: username,
  PASSWORD: password,
},
ClientId: clientId,
});

return client.send(command);
};
```

- Einzelheiten zur API finden Sie [InitiateAuth](#) in der AWS SDK for JavaScript API-Referenz.

ListUsers

Das folgende Codebeispiel zeigt die Verwendung `ListUsers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ListUsers](#) in der AWS SDK for JavaScript API-Referenz.

ResendConfirmationCode

Das folgende Codebeispiel zeigt die Verwendung `ResendConfirmationCode`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ResendConfirmationCode](#) in der AWS SDK for JavaScript API-Referenz.

RespondToAuthChallenge

Das folgende Codebeispiel zeigt die Verwendung `RespondToAuthChallenge`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
```

```
    code,
  }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new RespondToAuthChallengeCommand({
      ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
      ChallengeResponses: {
        SOFTWARE_TOKEN_MFA_CODE: code,
        USERNAME: username,
      },
      ClientId: clientId,
      UserPoolId: userPoolId,
      Session: session,
    });

    return client.send(command);
  };
};
```

- Einzelheiten zur API finden Sie [RespondToAuthChallenge](#) in der AWS SDK for JavaScript API-Referenz.

SignUp

Das folgende Codebeispiel zeigt die Verwendung `SignUp`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
```

```
UserAttributes: [{ Name: "email", Value: email }],
});

return client.send(command);
};
```

- Einzelheiten zur API finden Sie [SignUp](#) in der AWS SDK for JavaScript API-Referenz.

VerifySoftwareToken

Das folgende Codebeispiel zeigt die Verwendung `VerifySoftwareToken`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```


- Einzelheiten zur API finden Sie [VerifySoftwareToken](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Registrieren eines Benutzers bei einem Benutzerpool, der MFA erfordert

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Registrieren Sie einen Benutzer mit einem Benutzernamen, einem Passwort und einer E-Mail-Adresse und bestätigen Sie ihn.
- Einrichten der Multi-Faktor-Authentifizierung durch Zuordnung einer MFA-Anwendung zu dem Benutzer.
- Anmelden unter Verwendung eines Passworts und eines MFA-Codes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Um die beste Erfahrung zu erzielen, klonen Sie das GitHub Repository und führen Sie dieses Beispiel aus. Der folgende Code ist ein Teil der vollständigen Beispielanwendung.

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};
```

```
const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up'
command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_ , username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    log(`Signing up.`);
    await signUp({ clientId, username, password, email });
    log(`Signed up. A confirmation email has been sent to: ${email}.`);
    log(`Run 'confirm-sign-up ${username} <code>' to confirm your account.`);
  } catch (err) {
    log(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

```
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "../constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
  }
};
```

```
    log(`Confirming user.`);
    await confirmSignUp({ clientId, username, code });
    log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
in.`,
    );
  } catch (err) {
    log(err);
  }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../actions/associate-software-token.js";
import { FILE_USER_POOLS } from "../constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-token' to finish the setup.",
  );
  qrcode.generate(
```

```
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-initiate-auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);
  }
};
```

```
log("Signing in.");
const { ChallengeName, Session } = await adminInitiateAuth({
  clientId,
  userPoolId,
  username,
  password,
});

if (ChallengeName === "MFA_SETUP") {
  log("MFA setup is required.");
  return handleMfaSetup(Session, username);
}

if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
  handleSoftwareTokenMfa(Session);
  log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
}
} catch (err) {
  log(err);
}
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../actions/admin-respond-to-auth-challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
```

```
    if (!username) {
      throw new Error(
        `Username is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`
      );
    }
  };

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_ , username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    log("Successfully authenticated.");
  } catch (err) {
    log(err);
  }
};
```

```
    }  
  };  
  
  export { adminRespondToAuthChallengeHandler };  
  
  const respondToAuthChallenge = ({  
    clientId,  
    username,  
    session,  
    userPoolId,  
    code,  
  }) => {  
    const client = new CognitoIdentityProviderClient({});  
  
    const command = new RespondToAuthChallengeCommand({  
      ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,  
      ChallengeResponses: {  
        SOFTWARE_TOKEN_MFA_CODE: code,  
        USERNAME: username,  
      },  
      ClientId: clientId,  
      UserPoolId: userPoolId,  
      Session: session,  
    });  
  
    return client.send(command);  
  };  
  
  import { log } from "@aws-doc-sdk-examples/lib/utils/util-log.js";  
  import { verifySoftwareToken } from "../../../../../actions/verify-software-token.js";  
  
  const validateTotp = (totp) => {  
    if (!totp) {  
      throw new Error(  
        `Time-based one-time password (TOTP) must be provided to the 'validate-  
software-token' command.`,  
      );  
    }  
  };  
  
  const verifySoftwareTokenHandler = async (commands) => {  
    const [, totp] = commands;  
  
    try {  
      validateTotp(totp);  
    }  
  };  
};
```



```
    log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    console.log(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)

- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Amazon DocumentDB DocumentDB-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon DocumentDB Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen


- [Serverless-Beispiele](#)

Serverless-Beispiele

Rufen Sie eine Lambda-Funktion von einem Amazon DocumentDB-Trigger aus auf

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzlast ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden. 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));
};
```

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden 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));
};
```

DynamoDB-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit DynamoDB verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo DynamoDB

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit DynamoDB.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Weitere Informationen zur Arbeit mit DynamoDB finden Sie unter [DynamoDB programmieren](#) mit AWS SDK for JavaScript JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Einzelheiten zur API finden Sie unter [ListTables](#)API-Referenz.AWS SDK for JavaScript

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

BatchExecuteStatement

Das folgende Codebeispiel zeigt die VerwendungBatchExecuteStatement.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie mithilfe von PartiQL einen Stapel von Elementen.

```

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};

```

Rufen Sie mithilfe von PartiQL einen Stapel von Elementen ab.

```

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
      }
    ]
  });

```

```
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

```
};
```

Löschen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [BatchExecuteStatement](#) in der AWS SDK for JavaScript API-Referenz.

BatchGetItem

Das folgende Codebeispiel zeigt die Verwendung `BatchGetItem`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [BatchGet](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses["Books"]);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [BatchGetItem](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
}
```

```
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [BatchGetItem](#) in der AWS SDK for JavaScript API-Referenz.

BatchWriteItem

Das folgende Codebeispiel zeigt die Verwendung `BatchWriteItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [BatchWrite](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
```

```
`${dirname}../../../../../../../../resources/sample_files/movies.json`,
);

const movies = JSON.parse(file.toString());

// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);

// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      // An existing table is required. A composite key of 'title' and 'year' is
recommended
      // to account for duplicate titles.
      ["BatchWriteMoviesTable"]: putRequests,
    },
  });

  await docClient.send(command);
}
};
```

- Einzelheiten zur API finden Sie [BatchWriteItem](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
  {
    PutRequest: {
      Item: {
        KEY: { N: "KEY_VALUE" },
        ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
        ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
      },
    },
  },
],
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [BatchWriteItem](#) in der AWS SDK for JavaScript API-Referenz.

CreateTable

Das folgende Codebeispiel zeigt die Verwendung `CreateTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  const response = await client.send(command);
}
```

```
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateTable](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
```

```
    KeyType: "RANGE",
  },
],
ProvisionedThroughput: {
  ReadCapacityUnits: 1,
  WriteCapacityUnits: 1,
},
TableName: "CUSTOMER_LIST",
StreamSpecification: {
  StreamEnabled: false,
},
});

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateTable](#) in der AWS SDK for JavaScript API-Referenz.

DeleteItem

Das folgende Codebeispiel zeigt die Verwendung `DeleteItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [DeleteCommand](#).


```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteItem](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie ein Element aus einer Tabelle.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
```

```
    TableName: "TABLE",
    Key: {
      KEY_NAME: { N: "VALUE" },
    },
  };

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Löschen Sie ein Element mithilfe des DynamoDB-Dokument-Clients aus einer Tabelle.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [DeleteItem](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTable

Das folgende Codebeispiel zeigt die Verwendung `DeleteTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteTable](#) in der AWS SDK for JavaScript API-Referenz.

DescribeTable

Das folgende Codebeispiel zeigt die Verwendung `DescribeTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});
```

```
export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeTable](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

```
  }  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeTable](#) in der AWS SDK for JavaScript API-Referenz.

DescribeTimeToLive

Das folgende Codebeispiel zeigt die Verwendung `DescribeTimeToLive`.

SDK für JavaScript (v3)

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-dynamodb";  
  
const describeTableTTL = async (tableName, region) => {  
  const client = new DynamoDBClient({  
    region: region,  
    endpoint: `https://dynamodb.${region}.amazonaws.com`  
  });  
  
  try {  
    const ttlDescription = await client.send(new  
DescribeTimeToLiveCommand({ TableName: tableName }));  
  
    if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {  
      console.log("TTL is enabled for table %s.", tableName);  
    } else {  
      console.log("TTL is not enabled for table %s.", tableName);  
    }  
  
    return ttlDescription;  
  } catch (e) {  
    console.error(`Error describing table: ${e}`);  
    throw e;  
  }  
}  
  
// enter table name and change region if desired.
```

```
describeTableTTL('your-table-name', 'us-east-1');
```

- Einzelheiten zur API finden Sie [DescribeTimeToLive](#) in der AWS SDK for JavaScript API-Referenz.

ExecuteStatement

Das folgende Codebeispiel zeigt die Verwendung `ExecuteStatement`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Rufen Sie ein Element mithilfe von PartiQL ab.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });
};
```



```
const response = await docClient.send(command);
console.log(response);
return response;
};
```

Löschen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });


  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der AWS SDK for JavaScript API-Referenz.

GetItem

Das folgende Codebeispiel zeigt die Verwendung `GetItem`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [GetCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";


const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [GetItem](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie ein Element aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Rufen Sie ein Element mithilfe des DynamoDB-Dokument-Clients aus einer Tabelle ab.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetItem](#) in der AWS SDK for JavaScript API-Referenz.

ListTables

Das folgende Codebeispiel zeigt die Verwendung `ListTables`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTables](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTables](#) in der AWS SDK for JavaScript API-Referenz.

PutItem

Das folgende Codebeispiel zeigt die Verwendung `PutItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [PutCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [PutItem](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Fügen Sie ein Element in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
```

```
var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Fügen Sie ein Element mithilfe des DynamoDB-Dokument-Clients in eine Tabelle ein.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

```
}  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutItem](#) in der AWS SDK for JavaScript API-Referenz.

Query

Das folgende Codebeispiel zeigt die Verwendung Query.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [QueryCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";  
  
const client = new DynamoDBClient({});  
const docClient = DynamoDBDocumentClient.from(client);  
  
export const main = async () => {  
  const command = new QueryCommand({  
    TableName: "CoffeeCrop",  
    KeyConditionExpression:  
      "OriginCountry = :originCountry AND RoastDate > :roastDate",  
    ExpressionAttributeValues: {  
      ":originCountry": "Ethiopia",  
      ":roastDate": "2023-05-01",  
    },  
    ConsistentRead: true,  
  });  
  
  const response = await docClient.send(command);
```



```
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Query](#) in der AWS SDK for JavaScript -API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

```
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Query](#) in der AWS SDK for JavaScript -API-Referenz.

Scan

Das folgende Codebeispiel zeigt, wie man es benutztScan.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [ScanCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Weitere API-Informationen finden Sie unter [Scan](#) in der AWS SDK for JavaScript -API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
  // want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
```

```
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
    );
});
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Scan](#) in der AWS SDK for JavaScript -API-Referenz.

UpdateItem

Das folgende Codebeispiel zeigt, wie man es benutztUpdateItem.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [UpdateCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
```

```
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [UpdateItem](#) unter AWS SDK for JavaScript API-Referenz.

UpdateTimeToLive

Das folgende Codebeispiel zeigt die Verwendung `UpdateTimeToLive`.

SDK für JavaScript (v3)

Aktivieren Sie TTL für eine bestehende DynamoDB-Tabelle.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

  const client = new DynamoDBClient({});

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: true,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL enabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    }
  }
};
```

```

    } else {
      console.log(`Failed to enable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
    console.error(`Error enabling TTL: ${e}`);
    throw e;
  }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Deaktivieren Sie TTL für eine bestehende DynamoDB-Tabelle.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const disableTTL = async (tableName, ttlAttribute) => {

  const client = new DynamoDBClient({});

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: false,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL disabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    } else {
      console.log(`Failed to disable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
```

```
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Einzelheiten zur API finden Sie unter [UpdateTimeToLiveAPI-Referenz](#).AWS SDK for JavaScript

Szenarien

Aktualisieren Sie die TTL eines Artikels unter bestimmten Bedingungen

Das folgende Codebeispiel zeigt, wie die TTL eines Elements bedingt aktualisiert wird.

SDK für JavaScript (v3)

Aktualisieren Sie TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

const updateDynamoDBItem = async (tableName, region, partitionKey, sortKey,
  newAttribute) => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {
    TableName: tableName,
    Key: marshall({
      artist: partitionKey,
      album: sortKey
    }),
    UpdateExpression: "SET newAttribute = :newAttribute",
    ConditionExpression: "expireAt > :expiration",
    ExpressionAttributeValues: marshall({
```

```

        ':newAttribute': newAttribute,
        ':expiration': currentTime
    })),
    ReturnValues: "ALL_NEW"
};

try {
    const response = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(response.Attributes);
    console.log("Item updated successfully: ", responseData);
    return responseData;
} catch (error) {
    if (error.name === "ConditionalCheckFailedException") {
        console.log("Condition check failed: Item's 'expireAt' is expired.");
    } else {
        console.error("Error updating item: ", error);
    }
    throw error;
}
};

// Enter your values here
updateDynamoDBItem('your-table-name', "us-east-1", 'your-partition-key-value', 'your-
sort-key-value', 'your-new-attribute-value');

```

- Einzelheiten zur API finden Sie unter [UpdateItem](#)API-Referenz.AWS SDK for JavaScript

Erstellen Sie ein Objekt mit einer TTL

Das folgende Codebeispiel zeigt, wie ein Element mit TTL erstellt wird.

SDK für JavaScript (v3)

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });
}

```



```
// Get the current time in epoch second format
const current_time = Math.floor(new Date().getTime() / 1000);

// Calculate the expireAt time (90 days from now) in epoch second format
const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
1000);

// Create DynamoDB item
const item = {
  'partitionKey': {'S': partition_key},
  'sortKey': {'S': sort_key},
  'createdAt': {'N': current_time.toString()},
  'expireAt': {'N': expire_at.toString()}
};

const putItemCommand = new PutItemCommand({
  TableName: table_name,
  Item: item,
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
});

client.send(putItemCommand, function(err, data) {
  if (err) {
    console.log("Exception encountered when creating item %s, here's what
happened: ", data, ex);
    throw err;
  } else {
    console.log("Item created successfully: %s.", data);
    return data;
  }
});
}

// use your own values
createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
'your-sort-key-value');
```

- Einzelheiten zur API finden Sie [PutItem](#) in der AWS SDK for JavaScript API-Referenz.

Erste Schritte mit Tabellen, Elementen und Abfragen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen einer Tabelle, die Filmdaten enthalten kann.
- Einfügen, Abrufen und Aktualisieren eines einzelnen Films in der Tabelle.
- Schreiben von Filmdaten in die Tabelle anhand einer JSON-Beispieldatei.
- Abfragen nach Filmen, die in einem bestimmten Jahr veröffentlicht wurden.
- Scan nach Filmen, die in mehreren Jahren veröffentlicht wurden.
- Löschen eines Films aus der Tabelle und anschließendes Löschen der Tabelle.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { readFileSync } from "fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
  GetCommand,
  PutCommand,
  UpdateCommand,
```

```
    paginateQuery,
    paginateScan,
  } from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
  });
```

```
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
KeySchema: [
  // The way your data is accessed determines how you structure your keys.
  // The movies table will be queried for movies by year. It makes sense
  // to make year our partition (HASH) key.
  { AttributeName: "year", KeyType: "HASH" },
  { AttributeName: "title", KeyType: "RANGE" },
],
});

log("Creating a table.");
const createTableResponse = await client.send(createTableCommand);
log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Add a movie to the table.
 */

log("Adding a single movie to the table.");
// PutCommand is the first example usage of 'lib-dynamodb'.
const putCommand = new PutCommand({
  TableName: tableName,
  Item: {
    // In 'client-dynamodb', the AttributeValue would be required ( `year: { N:
1981 }` )
    // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
    year: 1981,
    // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
    // is required.
    title: "The Evil Dead",
    // Every other attribute is optional.
    info: {
      genres: ["Horror"],
    },
  },
});
await docClient.send(putCommand);
```

```
log("The movie was added.");

/**
 * Get a movie from the table.
 */

log("Getting a single movie from the table.");
const getCommand = new GetCommand({
  TableName: tableName,
  // Requires the complete primary key. For the movies table, the primary key
  // is only the id (partition key).
  Key: {
    year: 1981,
    title: "The Evil Dead",
  },
  // Set this to make sure that recent writes are reflected.
  // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html.
  ConsistentRead: true,
});
const getResponse = await docClient.send(getCommand);
log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

/**
 * Update a movie in the table.
 */

log("Updating a single movie in the table.");
const updateCommand = new UpdateCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
  // This update expression appends "Comedy" to the list of genres.
  // For more information on update expressions, see
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.UpdateExpressions.html
  UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
  ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
  ExpressionAttributeValues: {
    ":vals": ["Comedy"],
  },
  ReturnValues: "ALL_NEW",
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);
```

```
/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
log("Movies added.");

/**
```

```
* Query for movies by year.
*/

log("Querying for all movies from 1981.");
const paginatedQuery = paginateQuery(
  { client: docClient },
  {
    TableName: tableName,
    //For more information about query expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
    KeyConditionExpression: "#y = :y",
    // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
    // name by using an expression attribute name.
    ExpressionAttributeNames: { "#y": "year" },
    ExpressionAttributeValues: { ":y": 1981 },
    ConsistentRead: true,
  },
);
/**
 * @type { Record<string, any>[] };
 */
const movies1981 = [];
for await (const page of paginatedQuery) {
  movies1981.push(...page.Items);
}
log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

/**
 * Scan the table for movies between 1980 and 1990.
 */

log(`Scan for movies released between 1980 and 1990`);
// A 'Scan' operation always reads every item in the table. If your design
requires
// the use of 'Scan', consider indexing your table or changing your design.
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
const paginatedScan = paginateScan(
  { client: docClient },
  {
    TableName: tableName,
    // Scan uses a filter expression instead of a key condition expression. Scan
will
```

```
// read the entire table and then apply the filter.
FilterExpression: "#y between :y1 and :y2",
ExpressionAttributeNames: { "#y": "year" },
ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
ConsistentRead: true,
},
);
/**
 * @type { Record<string, any>[] };
 */
const movies1980to1990 = [];
for await (const page of paginatedScan) {
  movies1980to1990.push(...page.Items);
}
log(
  `Movies: ${movies1980to1990
    .map((m) => `${m.title} (${m.year})`)
    .join(", ")}`);
);

/**
 * Delete the table.
 */

const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
log(`Deleting table ${tableName}.`);
await client.send(deleteTableCommand);
log("Table deleted.");
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)
 - [GetItem](#)
 - [PutItem](#)
 - [Abfrage](#)

- [Scan](#)
- [UpdateItem](#)

Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Stapels von Elementen mithilfe mehrerer SELECT-Anweisungen.
- Hinzufügen eines Stapels von Elementen hinzu, indem mehrere INSERT-Anweisungen ausgeführt werden.
- Aktualisieren eines Stapels von Elementen mithilfe mehrerer UPDATE-Anweisungen.
- Löschen eines Stapels von Elementen mithilfe mehrerer DELETE-Anweisungen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie Batch-PartiQL-Anweisungen aus.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
```

```
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
```

```
Statements: [
  {
    Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
    Parameters: ["Alachua", 10712],
  },
  {
    Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
    Parameters: ["High Springs", 6415],
  },
],
});
await docClient.send(addItemsStatementCommand);
log(`Cities inserted.`);

/**
 * Select items.
 */

log("Selecting cities from the table.");
const selectItemsStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
  Statements: [
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
const selectItemResponse = await docClient.send(selectItemsStatementCommand);
log(
  `Got cities: ${selectItemResponse.Responses.map(
    (r) => `${r.Item.name} (${r.Item.population})`,
  )}.join(", ")`,
);

/**
 * Update items.
 */
```

```
log("Modifying the populations.");
const updateItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
  Statements: [
    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [10, "Alachua"],
    },
    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [5, "High Springs"],
    },
  ],
});
await docClient.send(updateItemStatementCommand);
log(`Updated cities.`);

/**
 * Delete the items.
 */

log("Deleting the cities.");
const deleteItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
  Statements: [
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
await docClient.send(deleteItemStatementCommand);
log("Cities deleted.");

/**
 * Delete the table.
 */
```

```
log("Deleting the table.");
const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
await client.send(deleteTableCommand);
log("Table deleted.");
};
```

- Einzelheiten zur API finden Sie [BatchExecuteStatement](#) in der AWS SDK for JavaScript API-Referenz.

Abfragen einer Tabelle mit PartiQL

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.
- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie einzelne PartiQL-Anweisungen aus.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
```

```
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }
}

/**
```

```
* Create a table.
*/

log("Creating a table.");
const createTableCommand = new CreateTableCommand({
  TableName: tableName,
  // This example performs a large write to the database.
  // Set the billing mode to PAY_PER_REQUEST to
  // avoid throttling the large write.
  BillingMode: BillingMode.PAY_PER_REQUEST,
  // Define the attributes that are necessary for the key schema.
  AttributeDefinitions: [
    {
      AttributeName: "varietal",
      // 'S' is a data type descriptor that represents a number type.
      // For a list of all data type descriptors, see the following link.
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
      Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
      AttributeType: "S",
    },
  ],
  // The KeySchema defines the primary key. The primary key can be
  // a partition key, or a combination of a partition key and a sort key.
  // Key schema design is important. For more info, see
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
  practices.html
  KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
});
await client.send(createTableCommand);
log(`Table created: ${tableName}.`);

/**
 * Wait until the table is active.
 */

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Insert an item.
 */
```



```
log("Inserting a coffee into the table.");
const addItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
  Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
  Parameters: ["arabica", ["chocolate", "floral"]],
});
await client.send(addItemStatementCommand);
log(`Coffee inserted.`);

/**
 * Select an item.
 */

log("Selecting the coffee from the table.");
const selectItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
  Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
  Parameters: ["arabica"],
});
const selectItemResponse = await docClient.send(selectItemStatementCommand);
log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

/**
 * Update the item.
 */

log("Add a flavor profile to the coffee.");
const updateItemStatementCommand = new ExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
  Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
varietal=?`,
  Parameters: [["fruity"], "arabica"],
});
await client.send(updateItemStatementCommand);
log(`Updated coffee`);

/**
 * Delete the item.
 */
```

```

    log("Deleting the coffee.");
    const deleteItemStatementCommand = new ExecuteStatementCommand({
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
      Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
      Parameters: ["arabica"],
    });
    await docClient.send(deleteItemStatementCommand);
    log("Coffee deleted.");

    /**
     * Delete the table.
     */

    log("Deleting the table.");
    const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
    await client.send(deleteTableCommand);
    log("Table deleted.");
  };

```

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der AWS SDK for JavaScript API-Referenz.

Fragen Sie nach TTL-Elementen ab

Das folgende Codebeispiel zeigt, wie TTL-Elemente abgefragt werden.

SDK für JavaScript (v3)

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

async function queryDynamoDBItems(tableName, region, primaryKey) {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {

```

```

    TableName: tableName,
    KeyConditionExpression: "#pk = :pk",
    FilterExpression: "#ea > :ea",
    ExpressionAttributeNames: {
      "#pk": "primaryKey",
      "#ea": "expireAt"
    },
    ExpressionAttributeValues: marshall({
      ":pk": primaryKey,
      ":ea": currentTime
    })
  });

  try {
    const { Items } = await client.send(new QueryCommand(params));
    Items.forEach(item => {
      console.log(unmarshall(item))
    });
    return Items;
  } catch (err) {
    console.error(`Error querying items: ${err}`);
    throw err;
  }
}

//enter your own values here
queryDynamoDBItems('your-table-name', 'your-partition-key-value');
```

- Weitere API-Informationen finden Sie unter [Query](#) in der AWS SDK for JavaScript -API-Referenz.

Aktualisieren Sie die TTL eines Elements

Das folgende Codebeispiel zeigt, wie die TTL eines Elements aktualisiert wird.

SDK für JavaScript (v3)

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";
```

```
async function updateDynamoDBItem(tableName, region, partitionKey, sortKey) {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);
  const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

  const params = {
    TableName: tableName,
    Key: marshall({
      partitionKey: partitionKey,
      sortKey: sortKey
    }),
    UpdateExpression: "SET updatedAt = :c, expireAt = :e",
    ExpressionAttributeValues: marshall({
      ":c": currentTime,
      ":e": expireAt
    }),
  };

  try {
    const data = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(data.Attributes);
    console.log("Item updated successfully: %s", responseData);
    return responseData;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
}

//enter your values here
updateDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
  'your-sort-key-value');
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK for JavaScript API-Referenz.

Serverless-Beispiele

Rufen Sie eine Lambda-Funktion von einem DynamoDB-Trigger aus auf

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzlast ab und protokolliert den Inhalt des Datensatzes.

SDK für (v3) JavaScript

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. 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)}`);
};
```

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. 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)}`);  
};
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Trigger

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von 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: [] };  
};
```

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";  
  
export const handler = async (event: DynamoDBStreamEvent):  
    Promise<DynamoDBBatchItemFailure[]> => {  
  
    const batchItemsFailures: DynamoDBBatchItemFailure[] = []  
    let curRecordSequenceNumber  
  
    for(const record of event.Records) {  
        curRecordSequenceNumber = record.dynamodb?.SequenceNumber  
  
        if(curRecordSequenceNumber) {  
            batchItemsFailures.push({  
                itemIdentifier: curRecordSequenceNumber  
            })  
        }  
    }  
  
    return batchItemsFailures  
}
```

Amazon EC2 EC2-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon EC2 Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Amazon EC2

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon EC2.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
      .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
      .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};
```


- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AllocateAddress

Das folgende Codebeispiel zeigt die Verwendung `AllocateAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { AllocateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new AllocateAddressCommand({});

  try {
    const { AllocationId, PublicIp } = await client.send(command);
    console.log("A new IP address has been allocated to your account:");
    console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
    console.log(
      "You can view your IP addresses in the AWS Management Console for Amazon EC2. Look under Network & Security > Elastic IPs",
    );
  } catch (err) {
    console.error(err);
  }
}
```

```
};
```

- Einzelheiten zur API finden Sie [AllocateAddress](#) in der AWS SDK for JavaScript API-Referenz.

AssociateAddress

Das folgende Codebeispiel zeigt die Verwendung `AssociateAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { AssociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  // You need to allocate an Elastic IP address before associating it with an
  // instance.
  // You can do that with the AllocateAddressCommand.
  const allocationId = "ALLOCATION_ID";
  // You need to create an EC2 instance before an IP address can be associated with
  // it.
  // You can do that with the RunInstancesCommand.
  const instanceId = "INSTANCE_ID";
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
      ${instanceId}.`,
      `The association ID is ${AssociationId}.`,
    );
  }
};
```

```
    );  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Einzelheiten zur API finden Sie [AssociateAddress](#) in der AWS SDK for JavaScript API-Referenz.

AuthorizeSecurityGroupIngress

Das folgende Codebeispiel zeigt die Verwendung `AuthorizeSecurityGroupIngress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { AuthorizeSecurityGroupIngressCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
// Grant permissions for a single IP address to ssh into instances  
// within the provided security group.  
export const main = async () => {  
  const command = new AuthorizeSecurityGroupIngressCommand({  
    // Replace with a security group ID from the AWS console or  
    // the DescribeSecurityGroupsCommand.  
    GroupId: "SECURITY_GROUP_ID",  
    IpPermissions: [  
      {  
        IpProtocol: "tcp",  
        FromPort: 22,  
        ToPort: 22,  
        // Replace 0.0.0.0 with the IP address to authorize.  
        // For more information on this notation, see  
        // https://en.wikipedia.org/wiki/Classless_Inter-  
        Domain_Routing#CIDR_notation  
        IpRanges: [{ CidrIp: "0.0.0.0/32" }],  
      }  
    ],  
  });  
};
```

```
    },  
  ],  
});  
  
try {  
  const { SecurityGroupRules } = await client.send(command);  
  console.log(JSON.stringify(SecurityGroupRules, null, 2));  
} catch (err) {  
  console.error(err);  
}  
};
```

- Einzelheiten zur API finden Sie [AuthorizeSecurityGroupIngress](#) in der AWS SDK for JavaScript API-Referenz.

CreateKeyPair

Das folgende Codebeispiel zeigt die Verwendung `CreateKeyPair`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateKeyPairCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Create a key pair in Amazon EC2.  
    const { KeyMaterial, KeyName } = await client.send(  
      // A unique name for the key pair. Up to 255 ASCII characters.  
      new CreateKeyPairCommand({ KeyName: "KEY_PAIR_NAME" }),  
    );  
    // This logs your private key. Be sure to save it.  
    console.log(KeyName);  
  }  
};
```

```
    console.log(KeyMaterial);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [CreateKeyPair](#) in der AWS SDK for JavaScript API-Referenz.

CreateLaunchTemplate

Das folgende Codebeispiel zeigt die Verwendung `CreateLaunchTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  })),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  })),
);
```

- Einzelheiten zur API finden Sie [CreateLaunchTemplate](#) in der AWS SDK for JavaScript API-Referenz.

CreateSecurityGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateSecurityGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateSecurityGroupCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new CreateSecurityGroupCommand({
    // Up to 255 characters in length. Cannot start with sg-.
    GroupName: "SECURITY_GROUP_NAME",
    // Up to 255 characters in length.
    Description: "DESCRIPTION",
  });

  try {
    const { GroupId } = await client.send(command);
    console.log(GroupId);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [CreateSecurityGroup](#) in der AWS SDK for JavaScript API-Referenz.

DeleteKeyPair

Das folgende Codebeispiel zeigt die Verwendung `DeleteKeyPair`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteKeyPairCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DeleteKeyPairCommand({
    KeyName: "KEY_PAIR_NAME",
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteKeyPair](#) in der AWS SDK for JavaScript API-Referenz.

DeleteLaunchTemplate

Das folgende Codebeispiel zeigt die Verwendung `DeleteLaunchTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
await client.send(  
  new DeleteLaunchTemplateCommand({  
    LaunchTemplateName: NAMES.launchTemplateName,  
  }),  
);
```

- Einzelheiten zur API finden Sie [DeleteLaunchTemplate](#) in der AWS SDK for JavaScript API-Referenz.

DeleteSecurityGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteSecurityGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteSecurityGroupCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  const command = new DeleteSecurityGroupCommand({  
    GroupId: "GROUP_ID",  
  });  
  
  try {
```



```
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteSecurityGroup](#) in der AWS SDK for JavaScript API-Referenz.

DescribeAddresses

Das folgende Codebeispiel zeigt die Verwendung `DescribeAddresses`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeAddressesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: ["ALLOCATION_ID"],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (err) {
    console.error(err);
  }
}
```

```
};
```

- Einzelheiten zur API finden Sie [DescribeAddresses](#) in der AWS SDK for JavaScript API-Referenz.

DescribeIamInstanceProfileAssociations

Das folgende Codebeispiel zeigt die Verwendung `DescribeIamInstanceProfileAssociations`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeIamInstanceProfileAssociations](#) in der AWS SDK for JavaScript API-Referenz.

DescribeImages

Das folgende Codebeispiel zeigt die Verwendung `DescribeImages`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { paginateDescribeImages } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first i386 image available for EC2 instances.
export const main = async () => {
  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
    just sugar for
    // the MaxResults property in the base command.
    { client, pageSize: 25 },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
      ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: ["x86_64"] }],
    },
  );

  try {
    const arm64Images = [];
    for await (const page of paginator) {
      if (page.Images.length) {
        arm64Images.push(...page.Images);
        // Once we have at least 1 result, we can stop.
        if (arm64Images.length >= 1) {
          break;
        }
      }
    }
    console.log(arm64Images);
  } catch (err) {
```

```
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeImages](#) in der AWS SDK for JavaScript API-Referenz.

DescribeInstanceTypes

Das folgende Codebeispiel zeigt die Verwendung `DescribeInstanceTypes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  paginateDescribeInstanceTypes,
  DescribeInstanceTypesCommand,
} from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List at least the first arm64 EC2 instance type available.
export const main = async () => {
  // The paginate function is a wrapper around the underlying command.
  const paginator = paginateDescribeInstanceTypes(
    // Without limiting the page size, this call can take a long time. pageSize is
    just sugar for
    // the MaxResults property in the underlying command.
    { client, pageSize: 25 },
    {
      Filters: [
        { Name: "processor-info.supported-architecture", Values: ["x86_64"] },
        { Name: "free-tier-eligible", Values: ["true"] },
      ],
    }
  );
};
```

```
try {
  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...page.InstanceTypes);

      // When we have at least 1 result, we can stop.
      if (instanceTypes.length >= 1) {
        break;
      }
    }
  }
  console.log(instanceTypes);
} catch (err) {
  console.error(err);
}
};
```

- Einzelheiten zur API finden Sie [DescribeInstanceTypes](#) in der AWS SDK for JavaScript API-Referenz.

DescribeInstances

Das folgende Codebeispiel zeigt die Verwendung `DescribeInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// List all of your EC2 instances running with x86_64 architecture that were
// launched this month.
```

```
export const main = async () => {
  const d = new Date();
  const year = d.getFullYear();
  const month = `0${d.getMonth() + 1}`.slice(-2);
  const launchTimePattern = `${year}-${month}-*`;
  const command = new DescribeInstancesCommand({
    Filters: [
      { Name: "architecture", Values: ["x86_64"] },
      { Name: "instance-state-name", Values: ["running"] },
      {
        Name: "launch-time",
        Values: [launchTimePattern],
      },
    ],
  });

  try {
    const { Reservations } = await client.send(command);
    const instanceList = Reservations.reduce((prev, current) => {
      return prev.concat(current.Instances);
    }, []);


    console.log(instanceList);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeInstances](#) in der AWS SDK for JavaScript API-Referenz.

DescribeKeyPairs

Das folgende Codebeispiel zeigt die Verwendung `DescribeKeyPairs`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeKeyPairsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeKeyPairsCommand({});


  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeKeyPairs](#) in der AWS SDK for JavaScript API-Referenz.

DescribeRegions

Das folgende Codebeispiel zeigt die Verwendung `DescribeRegions`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeRegionsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions true even the regions that require opt-in will be returned.
    AllRegions: true,
    // You can omit the Filters property if you want to get all regions.
    Filters: [
      {
        Name: "region-name",
        // You can specify multiple values for a filter.
        // You can also use '*' as a wildcard. This will return all
        // of the regions that start with `us-east-`.
        Values: ["ap-southeast-4"],
      },
    ],
  });


  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeRegions](#) in der AWS SDK for JavaScript API-Referenz.

DescribeSecurityGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeSecurityGroups`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeSecurityGroupsCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Log the details of a specific security group.
export const main = async () => {
  const command = new DescribeSecurityGroupsCommand({
    GroupIds: ["SECURITY_GROUP_ID"],
  });


  try {
    const { SecurityGroups } = await client.send(command);
    console.log(JSON.stringify(SecurityGroups, null, 2));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der AWS SDK for JavaScript API-Referenz.

DescribeSubnets

Das folgende Codebeispiel zeigt die Verwendung `DescribeSubnets`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeSubnets](#) in der AWS SDK for JavaScript API-Referenz.

DescribeVpcs

Das folgende Codebeispiel zeigt die Verwendung `DescribeVpcs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
const client = new EC2Client({});
const { Vpcs } = await client.send(
  new DescribeVpcsCommand({
    Filters: [{ Name: "is-default", Values: ["true"] }],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeVpcs](#) in der AWS SDK for JavaScript API-Referenz.

DisassociateAddress

Das folgende Codebeispiel zeigt die Verwendung `DisassociateAddress`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DisassociateAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Disassociate an Elastic IP address from an instance.
export const main = async () => {
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    // retired.
    AssociationId: "ASSOCIATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DisassociateAddress](#) in der AWS SDK for JavaScript API-Referenz.

MonitorInstances

Das folgende Codebeispiel zeigt die Verwendung `MonitorInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { MonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Turn on detailed monitoring for the selected instance.
// By default, metrics are sent to Amazon CloudWatch every 5 minutes.
// For a cost you can enable detailed monitoring which sends metrics every minute.
export const main = async () => {
  const command = new MonitorInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });


  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
        ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [MonitorInstances](#) in der AWS SDK for JavaScript API-Referenz.

RebootInstances

Das folgende Codebeispiel zeigt die Verwendung `RebootInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { RebootInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new RebootInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });


  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [RebootInstances](#) in der AWS SDK for JavaScript API-Referenz.

ReleaseAddress

Das folgende Codebeispiel zeigt die Verwendung `ReleaseAddress`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ReleaseAddressCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    // retired.
    AllocationId: "ALLOCATION_ID",
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [ReleaseAddress](#) in der AWS SDK for JavaScript API-Referenz.

ReplaceIamInstanceProfileAssociation

Das folgende Codebeispiel zeigt die Verwendung `ReplaceIamInstanceProfileAssociation`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
```

```
);
```

- Einzelheiten zur API finden Sie [ReplacelamInstanceProfileAssociation](#) in der AWS SDK for JavaScript API-Referenz.

RunInstances

Das folgende Codebeispiel zeigt die Verwendung `RunInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { RunInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

// Create a new EC2 instance.
export const main = async () => {
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: "KEY_PAIR_NAME",
    // Your security group.
    SecurityGroupIds: ["SECURITY_GROUP_ID"],
    // An x86_64 compatible image.
    ImageId: "ami-0001a0d1a04bfcc30",
    // An x86_64 compatible free-tier instance type.
    InstanceType: "t1.micro",
    // Ensure only 1 instance launches.
    MinCount: 1,
    MaxCount: 1,
  });

  try {
    const response = await client.send(command);
    console.log(response);
  }
}
```

```
    } catch (err) {  
      console.error(err);  
    }  
  };  
};
```

- Einzelheiten zur API finden Sie [RunInstances](#) in der AWS SDK for JavaScript API-Referenz.

StartInstances

Das folgende Codebeispiel zeigt die Verwendung `StartInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { StartInstancesCommand } from "@aws-sdk/client-ec2";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  const command = new StartInstancesCommand({  
    // Use DescribeInstancesCommand to find InstanceIds  
    InstanceIds: ["INSTANCE_ID"],  
  });  
  
  try {  
    const { StartingInstances } = await client.send(command);  
    const instanceIdList = StartingInstances.map(  
      (instance) => ` • ${instance.InstanceId}`,  
    );  
    console.log("Starting instances:");  
    console.log(instanceIdList.join("\n"));  
  } catch (err) {  
    console.error(err);  
  }  
};
```


- Einzelheiten zur API finden Sie [StartInstances](#) in der AWS SDK for JavaScript API-Referenz.

StopInstances

Das folgende Codebeispiel zeigt die Verwendung `StopInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { StopInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new StopInstancesCommand({
    // Use DescribeInstancesCommand to find InstanceIds
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [StopInstances](#) in der AWS SDK for JavaScript API-Referenz.

TerminateInstances

Das folgende Codebeispiel zeigt die Verwendung `TerminateInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { TerminateInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new TerminateInstancesCommand({
    InstanceIds: ["INSTANCE_ID"],
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [TerminateInstances](#) in der AWS SDK for JavaScript API-Referenz.

UnmonitorInstances

Das folgende Codebeispiel zeigt die Verwendung `UnmonitorInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";

import { client } from "../libs/client.js";

export const main = async () => {
  const command = new UnmonitorInstancesCommand({
    InstanceIds: ["i-09a3dfe7ae00e853f"],
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instanceMonitoringsList = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
        ${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instanceMonitoringsList.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [UnmonitorInstances](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
```

```
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
```

```
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
```

```
        AttributeType: "N",
      },
    ],
    KeySchema: [
      {
        AttributeName: "MediaType",
        KeyType: "HASH",
      },
      {
        AttributeName: "ItemId",
        KeyType: "RANGE",
      },
    ],
  })),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
}),
```



```
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );

  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
```

```
MESSAGES.createdInstancePolicy
  .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
  .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
}),
```

```

new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({

```

```
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
    })),
    );
  })),
  new ScenarioOutput(
    "addedRoleToInstanceProfile",
    MESSAGES.addedRoleToInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  ...initParamsSteps,
  new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
  new ScenarioAction("createLaunchTemplate", async () => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    const ssmClient = new SSMClient({});
    const { Parameter } = await ssmClient.send(
      new GetParameterCommand({
        Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      })),
    );
    const ec2Client = new EC2Client({});
    await ec2Client.send(
      new CreateLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
        LaunchTemplateData: {
          InstanceType: "t3.micro",
          ImageId: Parameter.Value,
          IamInstanceProfile: { Name: NAMES.instanceProfileName },
          UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
          ).toString("base64"),
          KeyName: NAMES.keyPairName,
        },
      })),
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  );
  })),
  new ScenarioOutput(
    "createdLaunchTemplate",
    MESSAGES.createdLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    ),
  ),
```

```

    ),
    new ScenarioOutput(
      "creatingAutoScalingGroup",
      MESSAGES.creatingAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      ),
    ),
  ),
  new ScenarioAction("createAutoScalingGroup", async (state) => {
    const ec2Client = new EC2Client({});
    const { AvailabilityZones } = await ec2Client.send(
      new DescribeAvailabilityZonesCommand({}),
    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {

```

```

    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),
  new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
  new ScenarioAction("getSubnets", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
    const client = new EC2Client({});
    const { Subnets } = await client.send(
      new DescribeSubnetsCommand({
        Filters: [
          { Name: "vpc-id", Values: [state.defaultVpc] },
          { Name: "availability-zone", Values: state.availabilityZoneNames },
          { Name: "default-for-az", Values: ["true"] },
        ],
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
    state.subnets = Subnets.map((subnet) => subnet.SubnetId);
  }),
  new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",

```

```
MESSAGES.creatingLoadBalancerTargetGroup.replace(
  "${TARGET_GROUP_NAME}",
  NAMES.loadBalancerTargetGroupName,
),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
  state.targetGroupProtocol = targetGroup.Protocol;
  state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
  "createdLoadBalancerTargetGroup",
  MESSAGES.createdLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
```

```
        Subnets: state.subnets,
    })),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
        new CreateListenerCommand({
            LoadBalancerArn: state.loadBalancerArn,
            Protocol: state.targetGroupProtocol,
            Port: state.targetGroupPort,
            DefaultActions: [
                { Type: "forward", TargetGroupArn: state.targetGroupArn },
            ],
        }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
    const listener = Listeners[0];
    state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
    ),
),
```



```

),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
  "verifyInboundPort",
  /**
   *
   * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
   */
  async (state) => {
    const client = new EC2Client({});
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({
        Filters: [{ Name: "group-name", Values: ["default"] }],
      }),
    );
    if (!SecurityGroups) {
      state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
    }
    state.defaultSecurityGroup = SecurityGroups[0];

    /**
     * @type {string}

```

```

    */
    const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
    state.myIp = ipResponse.trim();
    const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
      ({ IpRanges }) =>
        IpRanges.some(
          ({ CidrIp }) =>
            CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
        ),
    )
    .filter(({ IpProtocol }) => IpProtocol === "tcp")
    .filter(({ FromPort }) => FromPort === 80);

    state.myIpRules = myIpRules;
  },
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",
        JSON.stringify(state.myIpRules, null, 2),
      );
    } else {
      return MESSAGES.noIpRules;
    }
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return false;
    } else {
      return MESSAGES.noIpRules;
    }
  },
),

```

```
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    },
  ),
  new ScenarioOutput("addedInboundRule", (state) => {
    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    } else {
      return false;
    }
  }),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  })
}
```

```
    }),
    new ScenarioOutput("verifiedEndpoint", (state) => {
      if (state.verifyEndpointError) {
        console.error(state.verifyEndpointError);
      } else {
        return MESSAGES.verifiedEndpoint.replace(
          "${ENDPOINT_RESPONSE}",
          state.endpointResponse,
        );
      }
    }),
  ];
```

Erstellen Sie Schritte, um die Demo auszuführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
```

```
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    } else {
      throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
  },
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
```

```
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
    { preformatted: true },
  );

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
  balancing-v2').TargetHealthDescription[]}} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
```

```
    whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
    input: new ScenarioInput(
      "loadBalancerCheck",
      MESSAGES.demoLoadBalancerCheck,
      {
        type: "confirm",
      },
    ),
    output: getRecommendationResult,
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
```

```
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        })
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        })
      );
    }
  })
}
```



```

    }),
    new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
    ...statusSteps,
    new ScenarioInput(
      "badCredentialsConfirmation",
      MESSAGES.demoBadCredentialsConfirmation,
      { type: "confirm" },
    ),
    new ScenarioAction("badCredentialsExit", (state) => {
      if (!state.badCredentialsConfirmation) {
        process.exit();
      }
    }),
    new ScenarioAction("fixDynamoDBName", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: NAMES.tableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioAction(
      "badCredentials",
      /**
       * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
       * state
       */
      async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
          new DescribeAutoScalingGroupsCommand({
            AutoScalingGroupNames: [NAMES.autoScalingGroupName],
          }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        // snippet-start:
        [javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
          new DescribeIamInstanceProfileAssociationsCommand({

```

```
        Filters: [
          { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
        ],
      )),
    );
    // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
    state.instanceProfileAssociationId =
      IamInstanceProfileAssociations[0].AssociationId;
    // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      ec2Client.send(
        new ReplaceIamInstanceProfileAssociationCommand({
          AssociationId: state.instanceProfileAssociationId,
          IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
        })),
    ),
  );
  // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

  await ec2Client.send(
    new RebootInstancesCommand({
      InstanceIds: [state.targetInstance.InstanceId],
    })),
  );

  const ssmClient = new SSMClient({});
  await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
    const { InstanceInformationList } = await ssmClient.send(
      new DescribeInstanceInformationCommand({}),
    );

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
```

```

        new SendCommandCommand({
            InstanceIds: [state.targetInstance.InstanceId],
            DocumentName: "AWS-RunShellScript",
            Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
        }),
    );
},
),
new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
    */
    (state) =>
        MESSAGES.demoTestBadCredentials.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
),
loadBalancerLoop,
new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmHealthCheckKey,
            Value: "deep",
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,

```

```
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
  ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
});
```

```
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
```

```
const { Policy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyName: NAMES.ssmOnlyPolicyName,
    PolicyDocument: readFileSync(
      join(RESOURCES_PATH, "ssm_only_policy.json"),
    ),
  }),
);
await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        },
      ],
    }),
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  }),
);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

```
);  
// snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]  
await iamClient.send(  
  new AddRoleToInstanceProfileCommand({  
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,  
    RoleName: NAMES.ssmOnlyRoleName,  
  }),  
);  
  
return InstanceProfile;  
}
```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { unlinkSync } from "node:fs";  
  
import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";  
import {  
  EC2Client,  
  DeleteKeyPairCommand,  
  DeleteLaunchTemplateCommand,  
} from "@aws-sdk/client-ec2";  
import {  
  IAMClient,  
  DeleteInstanceProfileCommand,  
  RemoveRoleFromInstanceProfileCommand,  
  DeletePolicyCommand,  
  DeleteRoleCommand,  
  DetachRolePolicyCommand,  
  paginateListPolicies,  
} from "@aws-sdk/client-iam";  
import {  
  AutoScalingClient,  
  DeleteAutoScalingGroupCommand,  
  TerminateInstanceInAutoScalingGroupCommand,  
  UpdateAutoScalingGroupCommand,  
  paginateDescribeAutoScalingGroups,  
} from "@aws-sdk/client-auto-scaling";  
import {  
  DeleteLoadBalancerCommand,
```

```
    DeleteTargetGroupCommand,
    DescribeTargetGroupsCommand,
    ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
```



```
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName } ),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  })),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  })),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.detachPolicyFromRoleError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.` ,
        );
      } else {
        await client.send(
          new DetachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: policy.Arn,
          } ),
        );
      }
    } catch (e) {
      state.detachPolicyFromRoleError = e;
    }
  })
}
```

```
    }),
    new ScenarioOutput("detachedPolicyFromRole", (state) => {
      if (state.detachPolicyFromRoleError) {
        console.error(state.detachPolicyFromRoleError);
        return MESSAGES.detachPolicyFromRoleError
          .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
          .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
      } else {
        return MESSAGES.detachedPolicyFromRole
          .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
          .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
      }
    })),
    new ScenarioAction("deleteInstancePolicy", async (state) => {
      const client = new IAMClient({});
      const policy = await findPolicy(NAMES.instancePolicyName);

      if (!policy) {
        state.deletePolicyError = new Error(
          `Policy ${NAMES.instancePolicyName} not found.`
        );
      } else {
        return client.send(
          new DeletePolicyCommand({
            PolicyArn: policy.Arn,
          }),
        );
      }
    })),
    new ScenarioOutput("deletePolicyResult", (state) => {
      if (state.deletePolicyError) {
        console.error(state.deletePolicyError);
        return MESSAGES.deletePolicyError.replace(
          "${INSTANCE_POLICY_NAME}",
          NAMES.instancePolicyName,
        );
      } else {
        return MESSAGES.deletedPolicy.replace(
          "${INSTANCE_POLICY_NAME}",
          NAMES.instancePolicyName,
        );
      }
    })),
    new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
```

```
try {
  const client = new IAMClient({});
  await client.send(
    new RemoveRoleFromInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
} catch (e) {
  state.removeRoleFromInstanceProfileError = e;
}
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  } else {
```

```
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  } else {
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
```

```
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  } else {
    return MESSAGES.deletedAutoScalingGroup.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  } else {
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
```

```
// snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
const client = new ElasticLoadBalancingV2Client({});
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
await client.send(
  new DeleteLoadBalancerCommand({
    LoadBalancerArn: loadBalancer.LoadBalancerArn,
  }),
);
await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
  const lb = await findLoadBalancer(NAMES.loadBalancerName);
  if (lb) {
    throw new Error("Load balancer still exists.");
  }
});
// snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
} catch (e) {
  state.deleteLoadBalancerError = e;
}
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
  }

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
```

```
        client.send(
            new DeleteTargetGroupCommand({
                TargetGroupArn: TargetGroups[0].TargetGroupArn,
            }),
        ),
    );
} catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
}
// snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
            "${TARGET_GROUP_NAME}",
            NAMES.loadBalancerTargetGroupName,
        );
    } else {
        return MESSAGES.deletedLoadBalancerTargetGroup.replace(
            "${TARGET_GROUP_NAME}",
            NAMES.loadBalancerTargetGroupName,
        );
    }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new RemoveRoleFromInstanceProfileCommand({
                InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
                RoleName: NAMES.ssmOnlyRoleName,
            }),
        );
    } catch (e) {
        state.detachSsmOnlyRoleFromProfileError = e;
    }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
        console.error(state.detachSsmOnlyRoleFromProfileError);
        return MESSAGES.detachSsmOnlyRoleFromProfileError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
});
```

```
    } else {
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
  })),
  new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: ssmOnlyPolicy.Arn,
        })),
    );
    } catch (e) {
      state.detachSsmOnlyCustomRolePolicyError = e;
    }
  })),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
      return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
  })),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        })),
    );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  })
```



```
    })),
    new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
      if (state.detachSsmOnlyAWSRolePolicyError) {
        console.error(state.detachSsmOnlyAWSRolePolicyError);
        return MESSAGES.detachSsmOnlyAWSRolePolicyError
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
          .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
      } else {
        return MESSAGES.detachedSsmOnlyAWSRolePolicy
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
          .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
      }
    })),
    new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
      try {
        const iamClient = new IAMClient({});
        await iamClient.send(
          new DeleteInstanceProfileCommand({
            InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
          }),
        );
      } catch (e) {
        state.deleteSsmOnlyInstanceProfileError = e;
      }
    })),
    new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
      if (state.deleteSsmOnlyInstanceProfileError) {
        console.error(state.deleteSsmOnlyInstanceProfileError);
        return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
          "${INSTANCE_PROFILE_NAME}",
          NAMES.ssmOnlyInstanceProfileName,
        );
      } else {
        return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
          "${INSTANCE_PROFILE_NAME}",
          NAMES.ssmOnlyInstanceProfileName,
        );
      }
    })),
    new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
      try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
```

```
        new DeletePolicyCommand({
            PolicyArn: ssmOnlyPolicy.Arn,
        }),
    );
} catch (e) {
    state.deleteSsmOnlyPolicyError = e;
}
}),
new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
        console.error(state.deleteSsmOnlyPolicyError);
        return MESSAGES.deleteSsmOnlyPolicyError.replace(
            "${POLICY_NAME}",
            NAMES.ssmOnlyPolicyName,
        );
    } else {
        return MESSAGES.deletedSsmOnlyPolicy.replace(
            "${POLICY_NAME}",
            NAMES.ssmOnlyPolicyName,
        );
    }
}),
new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
        const iamClient = new IAMClient({});
        await iamClient.send(
            new DeleteRoleCommand({
                RoleName: NAMES.ssmOnlyRoleName,
            }),
        );
    } catch (e) {
        state.deleteSsmOnlyRoleError = e;
    }
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
        console.error(state.deleteSsmOnlyRoleError);
        return MESSAGES.deleteSsmOnlyRoleError.replace(
            "${ROLE_NAME}",
            NAMES.ssmOnlyRoleName,
        );
    } else {
        return MESSAGES.deletedSsmOnlyRole.replace(
            "${ROLE_NAME}",
        );
    }
}),
```

```
        NAMES.ssmOnlyRoleName,
    );
  }
  })),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    } else {
      console.log(err.name);
      throw err;
    }
  }
}

/**
 * @param {string} groupName
 */
```

```
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)

- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplaceIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Erste Schritte mit Instances

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Schlüsselpaar und eine Sicherheitsgruppe.
- Wählen Sie ein Amazon Machine Image (AMI) und einen kompatiblen Instance-Typ aus und erstellen Sie anschließend eine Instance.
- Halten Sie die Instance an und starten Sie sie neu.
- ~~Verknüpfen einer Elastic-IP-Adresse mit der Instance.~~

- Stellen Sie über SSH eine Verbindung zu Ihrer Instance her und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
import { mkdtempSync, writeFileSync, rmSync } from "fs";
import { tmpdir } from "os";
import { join } from "path";
import { get } from "http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DescribeInstancesCommand,
  DescribeKeyPairsCommand,
  DescribeSecurityGroupsCommand,
  DisassociateAddressCommand,
  EC2Client,
  paginateDescribeImages,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";
```

```
import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

const ec2Client = new EC2Client();
const ssmClient = new SSMClient();

const prompter = new Prompter();
const confirmMessage = "Continue?";
const tmpDirectory = mkdtempSync(join(tmpdir(), "ec2-scenario-tmp"));

const createKeyPair = async (keyPairName) => {
  // Create a key pair in Amazon EC2.
  const { KeyMaterial, KeyPairId } = await ec2Client.send(
    // A unique name for the key pair. Up to 255 ASCII characters.
    new CreateKeyPairCommand({ KeyName: keyPairName }),
  );

  // Save the private key in a temporary location.
  writeFileSync(`${tmpDirectory}/${keyPairName}.pem`, KeyMaterial, {
    mode: 0o400,
  });

  return KeyPairId;
};

const describeKeyPair = async (keyPairName) => {
  const command = new DescribeKeyPairsCommand({
    KeyNames: [keyPairName],
  });
  const { KeyPairs } = await ec2Client.send(command);
  return KeyPairs[0];
};

const createSecurityGroup = async (securityGroupName) => {
  const command = new CreateSecurityGroupCommand({
    GroupName: securityGroupName,
    Description: "A security group for the Amazon EC2 example.",
  });
  const { GroupId } = await ec2Client.send(command);
  return GroupId;
};
```

```
const allocateIpAddress = async () => {
  const command = new AllocateAddressCommand({});
  const { PublicIp, AllocationId } = await ec2Client.send(command);
  return { PublicIp, AllocationId };
};

const getLocalIpAddress = () => {
  return new Promise((res, rej) => {
    get("http://checkip.amazonaws.com", (response) => {
      let data = "";
      response.on("data", (chunk) => (data += chunk));
      response.on("end", () => res(data.trim()));
    }).on("error", (err) => {
      rej(err);
    });
  });
};

const authorizeSecurityGroupIngress = async (securityGroupId) => {
  const ipAddress = await getLocalIpAddress();
  const command = new AuthorizeSecurityGroupIngressCommand({
    GroupId: securityGroupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  await ec2Client.send(command);
  return ipAddress;
};

const describeSecurityGroup = async (securityGroupName) => {
  const command = new DescribeSecurityGroupsCommand({
    GroupNames: [securityGroupName],
  });
  const { SecurityGroups } = await ec2Client.send(command);

  return SecurityGroups[0];
};
```



```
const getAmznLinux2AMIs = async () => {
  const AMIs = [];
  for await (const page of paginateGetParametersByPath(
    {
      client: ssmClient,
    },
    { Path: "/aws/service/ami-amazon-linux-latest" },
  )) {
    page.Parameters.forEach((param) => {
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    });
  }

  const imageDetails = [];

  for await (const page of paginateDescribeImages(
    { client: ec2Client },
    { ImageIds: AMIs },
  )) {
    imageDetails.push(...(page.Images || []));
  }

  const choices = imageDetails.map((image, index) => ({
    name: `${image.ImageId} - ${image.Description}`,
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an image.",
    choices,
  });

  return imageDetails[selectedIndex];
};

/**
 * @param {import('@aws-sdk/client-ec2').Image} imageDetails
 */
```

```
const getCompatibleInstanceTypes = async (imageDetails) => {
  const paginator = paginateDescribeInstanceTypes(
    { client: ec2Client, pageSize: 25 },
    {
      Filters: [
        {
          Name: "processor-info.supported-architecture",
          Values: [imageDetails.Architecture],
        },
        { Name: "instance-type", Values: ["*.micro", "*.small"] },
      ],
    },
  );

  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...(page.InstanceTypes || []));
    }
  }

  const choices = instanceTypes.map((type, index) => ({
    name: `${type.InstanceType} - Memory:${type.MemoryInfo.SizeInMiB}`,
    value: index,
  }));

  /**
   * @type {number}
   */
  const selectedIndex = await prompter.select({
    message: "Select an instance type.",
    choices,
  });
  return instanceTypes[selectedIndex];
};

const runInstance = async ({
  keyPairName,
  securityGroupId,
  imageId,
  instanceType,
}) => {
  const command = new RunInstancesCommand({
```

```
    KeyName: keyPairName,
    SecurityGroupIds: [securityGroupId],
    ImageId: imageId,
    InstanceType: instanceType,
    MinCount: 1,
    MaxCount: 1,
  });

  const { Instances } = await ec2Client.send(command);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [Instances[0].InstanceId] },
  );
  return Instances[0].InstanceId;
};

const describeInstance = async (instanceId) => {
  const command = new DescribeInstancesCommand({
    InstanceIds: [instanceId],
  });

  const { Reservations } = await ec2Client.send(command);
  return Reservations[0].Instances[0];
};

const displaySSHConnectionInfo = ({ publicIp, keyPairName }) => {
  return `ssh -i ${tmpDirectory}/${keyPairName}.pem ec2-user@${publicIp}`;
};

const stopInstance = async (instanceId) => {
  const command = new StopInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(command);
  await waitUntilInstanceStopped(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};

const startInstance = async (instanceId) => {
  const startCommand = new StartInstancesCommand({ InstanceIds: [instanceId] });
  await ec2Client.send(startCommand);
  await waitUntilInstanceStatusOk(
    { client: ec2Client },
    { InstanceIds: [instanceId] },
  );
};
```

```
);
return await describeInstance(instanceId);
};

const associateAddress = async ({ allocationId, instanceId }) => {
  const command = new AssociateAddressCommand({
    AllocationId: allocationId,
    InstanceId: instanceId,
  });

  const { AssociationId } = await ec2Client.send(command);
  return AssociationId;
};

const disassociateAddress = async (associationId) => {
  const command = new DisassociateAddressCommand({
    AssociationId: associationId,
  });
  try {
    await ec2Client.send(command);
  } catch (err) {
    console.warn(
      `Failed to disassociated address with association id: ${associationId}`,
      err,
    );
  }
};

const releaseAddress = async (allocationId) => {
  const command = new ReleaseAddressCommand({
    AllocationId: allocationId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Address with allocation ID ${allocationId} released.\n`);
  } catch (err) {
    console.log(
      `Failed to release address with allocation id: ${allocationId}.`,
      err,
    );
  }
};
```

```
const restartInstance = async (instanceId) => {
  console.log("Stopping instance.");
  await stopInstance(instanceId);
  console.log("Instance stopped.");
  console.log("Starting instance.");
  const { PublicIpAddress } = await startInstance(instanceId);
  return PublicIpAddress;
};

const terminateInstance = async (instanceId) => {
  const command = new TerminateInstancesCommand({
    InstanceIds: [instanceId],
  });

  try {
    await ec2Client.send(command);
    await waitUntilInstanceTerminated(
      { client: ec2Client },
      { InstanceIds: [instanceId] },
    );
    console.log(`Instance with ID ${instanceId} terminated.\n`);
  } catch (err) {
    console.warn(`Failed to terminate instance ${instanceId}.`, err);
  }
};

const deleteSecurityGroup = async (securityGroupId) => {
  const command = new DeleteSecurityGroupCommand({
    GroupId: securityGroupId,
  });

  try {
    await ec2Client.send(command);
    console.log(`Security group ${securityGroupId} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete security group ${securityGroupId}.`, err);
  }
};

const deleteKeyPair = async (keyPairName) => {
  const command = new DeleteKeyPairCommand({
    KeyName: keyPairName,
  });
};
```

```
    try {
      await ec2Client.send(command);
      console.log(`Key pair ${keyPairName} deleted.\n`);
    } catch (err) {
      console.warn(`Failed to delete key pair ${keyPairName}.`, err);
    }
  }
};

const deleteTemporaryDirectory = () => {
  try {
    rmSync(tmpDirectory, { recursive: true });
    console.log(`Temporary directory ${tmpDirectory} deleted.\n`);
  } catch (err) {
    console.warn(`Failed to delete temporary directory ${tmpDirectory}.`, err);
  }
};

export const main = async () => {
  const keyPairName = "ec2-scenario-key-pair";
  const securityGroupName = "ec2-scenario-security-group";

  let securityGroupId, ipAllocationId, publicIp, instanceId, associationId;

  console.log(wrapText("Welcome to the Amazon EC2 basic usage scenario."));

  try {
    // Prerequisites
    console.log(
      "Before you launch an instance, you'll need a few things:",
      "\n - A Key Pair",
      "\n - A Security Group",
      "\n - An IP Address",
      "\n - An AMI",
      "\n - A compatible instance type",
      "\n\n I'll go ahead and take care of the first three, but I'll need your help\n\n for the rest.",
    );

    await prompter.confirm({ message: confirmMessage });

    await createKeyPair(keyPairName);
    securityGroupId = await createSecurityGroup(securityGroupName);
    const { PublicIp, AllocationId } = await allocateIpAddress();
    ipAllocationId = AllocationId;
  }
};
```

```
publicIp = PublicIp;
const ipAddress = await authorizeSecurityGroupIngress(securityGroupId);

const { KeyName } = await describeKeyPair(keyPairName);
const { GroupName } = await describeSecurityGroup(securityGroupName);
console.log(`# created the key pair ${KeyName}.\n`);
console.log(
  `# created the security group ${GroupName}`,
  `and allowed SSH access from ${ipAddress} (your IP).\n`,
);
console.log(`# allocated ${publicIp} to be used for your EC2 instance.\n`);

await prompter.confirm({ message: confirmMessage });

// Creating the instance
console.log(wrapText("Create the instance."));
console.log(
  "You get to choose which image you want. Select an amazon-linux-2 image from
the following:",
);
const imageDetails = await getAmznLinux2AMIs();
const instanceTypeDetails = await getCompatibleInstanceTypes(imageDetails);
console.log("Creating your instance. This can take a few seconds.");
instanceId = await runInstance({
  keyPairName,
  securityGroupId,
  imageId: imageDetails.ImageId,
  instanceType: instanceTypeDetails.InstanceType,
});
const instanceDetails = await describeInstance(instanceId);
console.log(`# instance ${instanceId}.\n`);
console.log(instanceDetails);
console.log(
  `
\nYou should now be able to SSH into your instance from another terminal:`,
  `
\n${displaySSHConnectionInfo({
  publicIp: instanceDetails.PublicIpAddress,
  keyPairName,
})}`,
);

await prompter.confirm({ message: confirmMessage });

// Understanding the IP address.
console.log(wrapText("Understanding the IP address."));
```

```
console.log(
  "When you stop and start an instance, the IP address will change. I'll restart
your",
  "instance for you. Notice how the IP address changes.",
);
const ipAddressAfterRestart = await restartInstance(instanceId);
console.log(
  `
Instance started. The IP address changed from
${instanceDetails.PublicIpAddress} to ${ipAddressAfterRestart}`,
  `
${displaySSHConnectionInfo({
  publicIp: ipAddressAfterRestart,
  keyPairName,
})}`
);
await prompter.confirm({ message: confirmMessage });
console.log(
  `If you want to the IP address to be static, you can associate an allocated`,
  `IP address to your instance. I allocated ${publicIp} for you earlier, and now
I'll associate it to your instance.`
);
associationId = await associateAddress({
  allocationId: ipAllocationId,
  instanceId,
});
console.log(
  "Done. Now you should be able to SSH using the new IP.\n",
  `${displaySSHConnectionInfo({ publicIp, keyPairName })}`
);
await prompter.confirm({ message: confirmMessage });
console.log(
  "I'll restart the server again so you can see the IP address remains the
same.",
);
const ipAddressAfterAssociated = await restartInstance(instanceId);
console.log(
  `Done. Here's your SSH info. Notice the IP address hasn't changed.`
  `
${displaySSHConnectionInfo({
  publicIp: ipAddressAfterAssociated,
  keyPairName,
})}`
);
await prompter.confirm({ message: confirmMessage });
} catch (err) {
  console.error(err);
}
```



```
} finally {
  // Clean up.
  console.log(wrapText("Clean up."));
  console.log("Now I'll clean up all of the stuff I created.");
  await prompter.confirm({ message: confirmMessage });
  console.log("Cleaning up. Some of these steps can take a bit of time.");
  await disassociateAddress(associationId);
  await terminateInstance(instanceId);
  await releaseAddress(ipAllocationId);
  await deleteSecurityGroup(securityGroupId);
  deleteTemporaryDirectory();
  await deleteKeyPair(keyPairName);
  console.log(
    "Done cleaning up. Thanks for staying until the end!",
    "If you have any feedback please use the feedback button in the docs",
    "or create an issue on GitHub.",
  );
}
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.

- [AllocateAddress](#)
- [AssociateAddress](#)
- [AuthorizeSecurityGroupIngress](#)
- [CreateKeyPair](#)
- [CreateSecurityGroup](#)
- [DeleteKeyPair](#)
- [DeleteSecurityGroup](#)
- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)

- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Elastic Load Balancing — Beispiele für Version 2 mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Elastic Load Balancing — Version 2 verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Elastic Load Balancing

Die folgenden Codebeispiele zeigen, wie Sie mit Elastic Load Balancing beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [DescribeLoadBalancers](#) in der AWS SDK for JavaScript API-Referenz.

Themen


- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreateListener

Das folgende Codebeispiel zeigt die Verwendung `CreateListener`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
const client = new ElasticLoadBalancingV2Client({});
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
```

- Einzelheiten zur API finden Sie [CreateListener](#) in der AWS SDK for JavaScript API-Referenz.

CreateLoadBalancer

Das folgende Codebeispiel zeigt die Verwendung `CreateLoadBalancer`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
```

```
    }),  
  );  
  state.loadBalancerDns = LoadBalancers[0].DNSName;  
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;  
  await waitUntilLoadBalancerAvailable(  
    { client },  
    { Names: [NAMES.loadBalancerName] },  
  );
```

- Einzelheiten zur API finden Sie [CreateLoadBalancer](#) in der AWS SDK for JavaScript API-Referenz.

CreateTargetGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateTargetGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});  
const { TargetGroups } = await client.send(  
  new CreateTargetGroupCommand({  
    Name: NAMES.loadBalancerTargetGroupName,  
    Protocol: "HTTP",  
    Port: 80,  
    HealthCheckPath: "/healthcheck",  
    HealthCheckIntervalSeconds: 10,  
    HealthCheckTimeoutSeconds: 5,  
    HealthyThresholdCount: 2,  
    UnhealthyThresholdCount: 2,  
    VpcId: state.defaultVpc,  
  })),  
);
```

- Einzelheiten zur API finden Sie [CreateTargetGroup](#) in der AWS SDK for JavaScript API-Referenz.

DeleteLoadBalancer

Das folgende Codebeispiel zeigt die Verwendung `DeleteLoadBalancer`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.


```
const client = new ElasticLoadBalancingV2Client({});
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
await client.send(
  new DeleteLoadBalancerCommand({
    LoadBalancerArn: loadBalancer.LoadBalancerArn,
  }),
);
await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
  const lb = await findLoadBalancer(NAMES.loadBalancerName);
  if (lb) {
    throw new Error("Load balancer still exists.");
  }
});
```

- Einzelheiten zur API finden Sie [DeleteLoadBalancer](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTargetGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteTargetGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
```

- Einzelheiten zur API finden Sie [DeleteTargetGroup](#) in der AWS SDK for JavaScript API-Referenz.

DescribeLoadBalancers

Das folgende Codebeispiel zeigt die Verwendung `DescribeLoadBalancers`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [DescribeLoadBalancers](#) in der AWS SDK for JavaScript API-Referenz.

DescribeTargetGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeTargetGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeTargetGroups](#) in der AWS SDK for JavaScript API-Referenz.

DescribeTargetHealth

Das folgende Codebeispiel zeigt die Verwendung `DescribeTargetHealth`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeTargetHealth](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
    parseScenarioArgs(scenarios);
  }
```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
```

```
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
      }),
      AttributeDefinitions: [
```

```

        {
            AttributeName: "MediaType",
            AttributeType: "S",
        },
        {
            AttributeName: "ItemId",
            AttributeType: "N",
        },
    ],
    KeySchema: [
        {
            AttributeName: "MediaType",
            KeyType: "HASH",
        },
        {
            AttributeName: "ItemId",
            KeyType: "RANGE",
        },
    ],
    }),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
    const client = new DynamoDBClient({});
    /**
     * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
     */
    const recommendations = JSON.parse(
        readFileSync(join(RESOURCES_PATH, "recommendations.json")),
    );

    return client.send(
        new BatchWriteItemCommand({
            RequestItems: {
                [NAMES.tableName]: recommendations.map((item) => ({

```

```
        PutRequest: { Item: item },
      })),
    },
  })),
);
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );
  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
});
```

```
    ),
  })),
);
state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  ),
});
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
```



```
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
```

```
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
    ),
    new ScenarioAction("addRoleToInstanceProfile", () => {
        const client = new IAMClient({});
        return client.send(
            new AddRoleToInstanceProfileCommand({
                RoleName: NAMES.instanceRoleName,
                InstanceProfileName: NAMES.instanceProfileName,
            }),
        );
    }),
    new ScenarioOutput(
        "addedRoleToInstanceProfile",
        MESSAGES.addedRoleToInstanceProfile
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
    ),
    ...initParamsSteps,
    new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
    new ScenarioAction("createLaunchTemplate", async () => {
        // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
        const ssmClient = new SSMClient({});
        const { Parameter } = await ssmClient.send(
            new GetParameterCommand({
                Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
            }),
        );
        const ec2Client = new EC2Client({});
        await ec2Client.send(
            new CreateLaunchTemplateCommand({
                LaunchTemplateName: NAMES.launchTemplateName,
                LaunchTemplateData: {
                    InstanceType: "t3.micro",
                    ImageId: Parameter.Value,
                    IamInstanceProfile: { Name: NAMES.instanceProfileName },
                    UserData: readFileSync(
                        join(RESOURCES_PATH, "server_startup_script.sh"),
                    ).toString("base64"),
                    KeyName: NAMES.keyPairName,
                },
            }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
    });
}),
```

```

new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
      }),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)

```

```

        .replace(
            "${AVAILABILITY_ZONE_NAMES}",
            state.availabilityZoneNames.join(", "),
        ),
    ),
    new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
        type: "confirm",
    }),
    new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
    new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
    new ScenarioAction("getVpc", async (state) => {
        // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
        const client = new EC2Client({});
        const { Vpcs } = await client.send(
            new DescribeVpcsCommand({
                Filters: [{ Name: "is-default", Values: ["true"] }],
            }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
        state.defaultVpc = Vpcs[0].VpcId;
    }),
    new ScenarioOutput("gotVpc", (state) =>
        MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
    ),
    new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
    new ScenarioAction("getSubnets", async (state) => {
        // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
        const client = new EC2Client({});
        const { Subnets } = await client.send(
            new DescribeSubnetsCommand({
                Filters: [
                    { Name: "vpc-id", Values: [state.defaultVpc] },
                    { Name: "availability-zone", Values: state.availabilityZoneNames },
                    { Name: "default-for-az", Values: ["true"] },
                ],
            }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
        state.subnets = Subnets.map((subnet) => subnet.SubnetId);
    }),
    new ScenarioOutput(
        "gotSubnets",
        /**
         * @param {{ subnets: string[] }} state

```

```
    */
    (state) =>
      MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
  ),
  new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new CreateTargetGroupCommand({
        Name: NAMES.loadBalancerTargetGroupName,
        Protocol: "HTTP",
        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
```

```
new ScenarioAction("createLoadBalancer", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
    }),
  );
  state.loadBalancerDns = LoadBalancers[0].DNSName;
  state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
  await waitUntilLoadBalancerAvailable(
    { client },
    { Names: [NAMES.loadBalancerName] },
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
  MESSAGES.createdLoadBalancer
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioOutput(
  "creatingListener",
  MESSAGES.creatingLoadBalancerListener
    .replace("${LB_NAME}", NAMES.loadBalancerName)
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
),
new ScenarioAction("createListener", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
  const client = new ElasticLoadBalancingV2Client({});
  const { Listeners } = await client.send(
    new CreateListenerCommand({
      LoadBalancerArn: state.loadBalancerArn,
      Protocol: state.targetGroupProtocol,
      Port: state.targetGroupPort,
      DefaultActions: [
        { Type: "forward", TargetGroupArn: state.targetGroupArn },
      ],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
  const listener = Listeners[0];
  state.loadBalancerListenerArn = listener.ListenerArn;
```

```

    }),
    new ScenarioOutput("createdListener", (state) =>
      MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
      ),
    ),
    new ScenarioOutput(
      "attachingLoadBalancerTargetGroup",
      MESSAGES.attachingLoadBalancerTargetGroup
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
    ),
    new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
      // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
      const client = new AutoScalingClient({});
      await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          TargetGroupARNs: [state.targetGroupArn],
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
    }),
    new ScenarioOutput(
      "attachedLoadBalancerTargetGroup",
      MESSAGES.attachedLoadBalancerTargetGroup,
    ),
    new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
    new ScenarioAction(
      "verifyInboundPort",
      /**
       *
       * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
       */
      async (state) => {
        const client = new EC2Client({});
        const { SecurityGroups } = await client.send(
          new DescribeSecurityGroupsCommand({
            Filters: [{ Name: "group-name", Values: ["default"] }],
          }),
        );
      );
      if (!SecurityGroups) {

```

```

    state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
  }
  state.defaultSecurityGroup = SecurityGroups[0];

  /**
   * @type {string}
   */
  const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
  state.myIp = ipResponse.trim();
  const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
    ({ IpRanges }) =>
      IpRanges.some(
        ({ CidrIp }) =>
          CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
        ),
    )
    .filter(({ IpProtocol }) => IpProtocol === "tcp")
    .filter(({ FromPort }) => FromPort === 80);

  state.myIpRules = myIpRules;
},
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",
        JSON.stringify(state.myIpRules, null, 2),
      );
    } else {
      return MESSAGES.noIpRules;
    }
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {

```



```

    if (state.myIpRules.length > 0) {
      return false;
    } else {
      return MESSAGES.noIpRules;
    }
  },
  { type: "confirm" },
),
new ScenarioAction(
  "addInboundRule",
  /**
   * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
   */
  async (state) => {
    if (!state.shouldAddInboundRule) {
      return;
    }

    const client = new EC2Client({});
    await client.send(
      new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.defaultSecurityGroup.GroupId,
        CidrIp: `${state.myIp}/32`,
        FromPort: 80,
        ToPort: 80,
        IpProtocol: "tcp",
      }),
    );
  },
),
new ScenarioOutput("addedInboundRule", (state) => {
  if (state.shouldAddInboundRule) {
    return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
  } else {
    return false;
  }
}),
new ScenarioOutput("verifyingEndpoint", (state) =>
  MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioAction("verifyEndpoint", async (state) => {
  try {
    const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>

```

```
    axios.get(`http://${state.loadBalancerDns}`),
  );
  state.endpointResponse = JSON.stringify(response.data, null, 2);
} catch (e) {
  state.verifyEndpointError = e;
}
}),
new ScenarioOutput("verifiedEndpoint", (state) => {
  if (state.verifyEndpointError) {
    console.error(state.verifyEndpointError);
  } else {
    return MESSAGES.verifiedEndpoint.replace(
      "${ENDPOINT_RESPONSE}",
      state.endpointResponse,
    );
  }
}),
];
```

Erstellen Sie Schritte, um die Demo auszuführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
```

```
    CreateRoleCommand,  
    AttachRolePolicyCommand,  
    CreateInstanceProfileCommand,  
    AddRoleToInstanceProfileCommand,  
    waitUntilInstanceProfileExists,  
  } from "@aws-sdk/client-iam";  
import {  
  AutoScalingClient,  
  DescribeAutoScalingGroupsCommand,  
  TerminateInstanceInAutoScalingGroupCommand,  
} from "@aws-sdk/client-auto-scaling";  
import {  
  DescribeIamInstanceProfileAssociationsCommand,  
  EC2Client,  
  RebootInstancesCommand,  
  ReplaceIamInstanceProfileAssociationCommand,  
} from "@aws-sdk/client-ec2";  
  
import {  
  ScenarioAction,  
  ScenarioInput,  
  ScenarioOutput,  
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";  
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";  
  
import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";  
import { findLoadBalancer } from "./shared.js";  
  
const getRecommendation = new ScenarioAction(  
  "getRecommendation",  
  async (state) => {  
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);  
    if (loadBalancer) {  
      state.loadBalancerDnsName = loadBalancer.DNSName;  
      try {  
        state.recommendation = (  
          await axios.get(`http://${state.loadBalancerDnsName}`)  
        ).data;  
      } catch (e) {  
        state.recommendation = e instanceof Error ? e.message : e;  
      }  
    } else {  
      throw new Error(MESSAGES.demoFindLoadBalancerError);  
    }  
  }  
);
```

```
    },
  );

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-balancing-v2').TargetHealthDescription[] }} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);
```

```
const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
```

```
new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
...statusSteps,
new ScenarioInput(
  "brokenDependencyConfirmation",
  MESSAGES.demoBrokenDependencyConfirmation,
  { type: "confirm" },
),
new ScenarioAction("brokenDependency", async (state) => {
  if (!state.brokenDependencyConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    state.badTableName = `fake-table-${Date.now()}`;
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: state.badTableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testBrokenDependency", (state) =>
  MESSAGES.demoTestBrokenDependency.replace(
    "${TABLE_NAME}",
    state.badTableName,
  ),
),
...statusSteps,
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
```

```

        Value: "static",
        Overwrite: true,
        Type: "String",
    })),
    );
}
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmTableNameKey,
            Value: NAMES.tableName,
            Overwrite: true,
            Type: "String",
        })),
    );
}),
new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
     */
    async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
            new DescribeAutoScalingGroupsCommand({
                AutoScalingGroupNames: [NAMES.autoScalingGroupName],
            })),
        );
    });

```

```
state.targetInstance = AutoScalingGroups[0].Instances[0];
// snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
// snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
// snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);
// snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );

  const instance = InstanceInformationList.find(
    (info) => info.InstanceId === state.targetInstance.InstanceId,
  );
};
```



```

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
   */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,

```

```

        Type: "String",
    })),
    );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    (state) =>
        MESSAGES.demoKillInstanceConfirmation.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
    { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
        process.exit();
    }
}),
new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    async (state) => {
        const client = new AutoScalingClient({});
        await client.send(
            new TerminateInstanceInAutoScalingGroupCommand({
                InstanceId: state.targetInstance.InstanceId,
                ShouldDecrementDesiredCapacity: false,
            }),
        );
    },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,

```

```
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
```

```
healthCheckLoop,
loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  // snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
```

```

        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    })),
    );
    await waitUntilInstanceProfileExists(
        { client: iamClient },
        { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
    await iamClient.send(
        new AddRoleToInstanceProfileCommand({
            InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
            RoleName: NAMES.ssmOnlyRoleName,
        })),
    );

    return InstanceProfile;
}

```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
    EC2Client,
    DeleteKeyPairCommand,
    DeleteLaunchTemplateCommand,
} from "@aws-sdk/client-ec2";
import {
    IAMClient,
    DeleteInstanceProfileCommand,
    RemoveRoleFromInstanceProfileCommand,
    DeletePolicyCommand,
    DeleteRoleCommand,
    DetachRolePolicyCommand,
    paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
    AutoScalingClient,
    DeleteAutoScalingGroupCommand,

```

```
    TerminateInstanceInAutoScalingGroupCommand,
    UpdateAutoScalingGroupCommand,
    paginateDescribeAutoScalingGroups,
  } from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
  });
];
```

```
    );
  } else {
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }
}),
new ScenarioAction("deleteKeyPair", async (state) => {
  try {
    const client = new EC2Client({});
    await client.send(
      new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
    );
    unlinkSync(`${NAMES.keyPairName}.pem`);
  } catch (e) {
    state.deleteKeyPairError = e;
  }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
  if (state.deleteKeyPairError) {
    console.error(state.deleteKeyPairError);
    return MESSAGES.deleteKeyPairError.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  } else {
    return MESSAGES.deletedKeyPair.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
  try {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.detachPolicyFromRoleError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`
      );
    } else {
      await client.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.instanceRoleName,
          PolicyArn: policy.Arn,
        })
      );
    }
  }
})
}
```

```
    }),
    );
  }
} catch (e) {
  state.detachPolicyFromRoleError = e;
}
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
  if (state.detachPolicyFromRoleError) {
    console.error(state.detachPolicyFromRoleError);
    return MESSAGES.detachPolicyFromRoleError
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.deletePolicyError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`
    );
  } else {
    return client.send(
      new DeletePolicyCommand({
        PolicyArn: policy.Arn,
      }),
    );
  }
}),
new ScenarioOutput("deletePolicyResult", (state) => {
  if (state.deletePolicyError) {
    console.error(state.deletePolicyError);
    return MESSAGES.deletePolicyError.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
```



```
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  );
}
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
```

```
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  } else {
    return MESSAGES.deletedInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  } else {
    return MESSAGES.deletedInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
```

```
        await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
} catch (e) {
    state.deleteAutoScalingGroupError = e;
}
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
        console.error(state.deleteAutoScalingGroupError);
        return MESSAGES.deleteAutoScalingGroupError.replace(
            "${AUTO_SCALING_GROUP_NAME}",
            NAMES.autoScalingGroupName,
        );
    } else {
        return MESSAGES.deletedAutoScalingGroup.replace(
            "${AUTO_SCALING_GROUP_NAME}",
            NAMES.autoScalingGroupName,
        );
    }
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
        // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
        await client.send(
            new DeleteLaunchTemplateCommand({
                LaunchTemplateName: NAMES.launchTemplateName,
            }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
        state.deleteLaunchTemplateError = e;
    }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
    if (state.deleteLaunchTemplateError) {
        console.error(state.deleteLaunchTemplateError);
        return MESSAGES.deleteLaunchTemplateError.replace(
            "${LAUNCH_TEMPLATE_NAME}",
            NAMES.launchTemplateName,
        );
    } else {
        return MESSAGES.deletedLaunchTemplate.replace(
            "${LAUNCH_TEMPLATE_NAME}",

```

```
        NAMES.launchTemplateName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
```

```
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
// snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
```

```
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  } else {
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      })
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  } else {
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
```

```
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
      })),
    );
  } catch (e) {
    state.detachSsmOnlyAWSRolePolicyError = e;
  }
})),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
  if (state.detachSsmOnlyAWSRolePolicyError) {
    console.error(state.detachSsmOnlyAWSRolePolicyError);
    return MESSAGES.detachSsmOnlyAWSRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  } else {
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }
})),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      })),
    );
  } catch (e) {
    state.deleteSsmOnlyInstanceProfileError = e;
  }
})),
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
    console.error(state.deleteSsmOnlyInstanceProfileError);
    return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }
}
```

```
    }),
    new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
      try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
          new DeletePolicyCommand({
            PolicyArn: ssmOnlyPolicy.Arn,
          }),
        );
      } catch (e) {
        state.deleteSsmOnlyPolicyError = e;
      }
    }),
    new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
      if (state.deleteSsmOnlyPolicyError) {
        console.error(state.deleteSsmOnlyPolicyError);
        return MESSAGES.deleteSsmOnlyPolicyError.replace(
          "${POLICY_NAME}",
          NAMES.ssmOnlyPolicyName,
        );
      } else {
        return MESSAGES.deletedSsmOnlyPolicy.replace(
          "${POLICY_NAME}",
          NAMES.ssmOnlyPolicyName,
        );
      }
    }),
    new ScenarioAction("deleteSsmOnlyRole", async (state) => {
      try {
        const iamClient = new IAMClient({});
        await iamClient.send(
          new DeleteRoleCommand({
            RoleName: NAMES.ssmOnlyRoleName,
          }),
        );
      } catch (e) {
        state.deleteSsmOnlyRoleError = e;
      }
    }),
    new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
      if (state.deleteSsmOnlyRoleError) {
        console.error(state.deleteSsmOnlyRoleError);
        return MESSAGES.deleteSsmOnlyRoleError.replace(
```



```
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
    );
} else {
    return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
    );
}
}),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
    const client = new IAMClient({});
    const paginatedPolicies = paginateListPolicies({ client }, {});
    for await (const page of paginatedPolicies) {
        const policy = page.Policies.find((p) => p.PolicyName === policyName);
        if (policy) {
            return policy;
        }
    }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
    const client = new AutoScalingClient({});
    try {
        await client.send(
            new DeleteAutoScalingGroupCommand({
                AutoScalingGroupName: groupName,
            }),
        );
    } catch (err) {
        if (!(err instanceof Error)) {
            throw err;
        } else {
            console.log(err.name);
            throw err;
        }
    }
}
```

```
    }
  }

  /**
   * @param {string} groupName
   */
  async function terminateGroupInstances(groupName) {
    const autoScalingClient = new AutoScalingClient({});
    const group = await findAutoScalingGroup(groupName);
    await autoScalingClient.send(
      new UpdateAutoScalingGroupCommand({
        AutoScalingGroupName: group.AutoScalingGroupName,
        MinSize: 0,
      }),
    );
    for (const i of group.Instances) {
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        autoScalingClient.send(
          new TerminateInstanceInAutoScalingGroupCommand({
            InstanceId: i.InstanceId,
            ShouldDecrementDesiredCapacity: true,
          }),
        ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplaceIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

EventBridge Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden EventBridge.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt, wie man es benutztPutEvents.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});
```

```
const response = await client.send(
  new PutEventsCommand({
    Entries: [
      {
        Detail: JSON.stringify({ greeting: "Hello there." }),
        DetailType: detailType,
        Resources: resources,
        Source: source,
      },
    ],
  }),
);

console.log("PutEvents response:");
console.log(response);
// PutEvents response:
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],
//   FailedEntryCount: 0
// }

return response;
};
```

- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

ebevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK for JavaScript API-Referenz.

PutRule

Das folgende Codebeispiel zeigt die Verwendung `PutRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
EventBridgeTestRule-1696280037720'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

ebevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK for JavaScript API-Referenz.

PutTargets

Das folgende Codebeispiel zeigt die Verwendung `PutTargets`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//     totalRetryDelay: 0
//   },
//   FailedEntries: [],
//   FailedEntryCount: 0
// }

return response;
};
```

- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var ebevents = new AWS.EventBridge({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myEventBridgeTarget",
    },
  ],
};

ebevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

```
}  
});
```

- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK for JavaScript API-Referenz.

AWS Glue Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden AWS Glue.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Glue

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Glue beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";  
  
const client = new GlueClient({});
```

```
export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreateCrawler

Das folgende Codebeispiel zeigt die Verwendung `CreateCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
```

```
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateCrawler](#) in der AWS SDK for JavaScript API-Referenz.

CreateJob

Das folgende Codebeispiel zeigt die Verwendung `CreateJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateJob](#) in der AWS SDK for JavaScript API-Referenz.

DeleteCrawler

Das folgende Codebeispiel zeigt die Verwendung `DeleteCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteCrawler](#) in der AWS SDK for JavaScript API-Referenz.

DeleteDatabase

Das folgende Codebeispiel zeigt die Verwendung `DeleteDatabase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const deleteDatabase = (databaseName) => {
```

```
const client = new GlueClient({});

const command = new DeleteDatabaseCommand({
  Name: databaseName,
});

return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteDatabase](#) in der AWS SDK for JavaScript API-Referenz.

DeleteJob

Das folgende Codebeispiel zeigt die Verwendung `DeleteJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteJob](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTable

Das folgende Codebeispiel zeigt die Verwendung `DeleteTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der AWS SDK for JavaScript API-Referenz.

GetCrawler

Das folgende Codebeispiel zeigt die Verwendung `GetCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
```



```
});  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetCrawler](#) in der AWS SDK for JavaScript API-Referenz.

GetDatabase

Das folgende Codebeispiel zeigt die Verwendung `GetDatabase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getDatabase = (name) => {  
    const client = new GlueClient({});  
  
    const command = new GetDatabaseCommand({  
        Name: name,  
    });  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetDatabase](#) in der AWS SDK for JavaScript API-Referenz.

GetDatabases

Das folgende Codebeispiel zeigt die Verwendung `GetDatabases`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetDatabases](#) in der AWS SDK for JavaScript API-Referenz.

GetJob

Das folgende Codebeispiel zeigt die Verwendung `GetJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetJob](#) in der AWS SDK for JavaScript API-Referenz.

GetJobRun

Das folgende Codebeispiel zeigt die Verwendung `GetJobRun`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetJobRun](#) in der AWS SDK for JavaScript API-Referenz.

GetJobRuns

Das folgende Codebeispiel zeigt die Verwendung `GetJobRuns`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetJobRuns](#) in der AWS SDK for JavaScript API-Referenz.

GetTables

Das folgende Codebeispiel zeigt die Verwendung `GetTables`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetTables](#) in der AWS SDK for JavaScript API-Referenz.

ListJobs

Das folgende Codebeispiel zeigt die Verwendung `ListJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der AWS SDK for JavaScript API-Referenz.

StartCrawler

Das folgende Codebeispiel zeigt die Verwendung `StartCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

```
};
```

- Einzelheiten zur API finden Sie [StartCrawler](#) in der AWS SDK for JavaScript API-Referenz.

StartJobRun

Das folgende Codebeispiel zeigt die Verwendung `StartJobRun`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [StartJobRun](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erste Schritte mit Crawlern und Aufträgen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Crawler, der einen öffentlichen Amazon-S3-Bucket crawlt und eine Datenbank mit CSV-formatierten Metadaten generiert.
- Führen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf AWS Glue Data Catalog.
- Erstellen Sie einen Auftrag, um CSV-Daten aus dem S3-Bucket zu extrahieren, die Daten umzuwandeln und die JSON-formatierte Ausgabe in einen anderen S3-Bucket zu laden.
- Listen Sie Informationen zu Auftragsausführungen auf, zeigen Sie transformierte Daten an und bereinigen Sie Ressourcen.

Weitere Informationen finden Sie unter [Tutorial: Erste Schritte mit AWS Glue Studio](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie einen Crawler und führen Sie ihn aus, der einen öffentlichen Amazon Simple Storage Service (Amazon S3)-Bucket crawlt und eine Metadatendatenbank generiert, die die gefundenen CSV-formatierten Daten beschreibt.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});
```

```
const command = new GetCrawlerCommand({
  Name: name,
});

return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

/**
 * @param {{ createCrawler: import('.././../actions/create-crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }
};
```



```
    }

    return { ...context };
  };

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
};
```

Listen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-glue').GetTablesCommandOutput>}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
};
```

Erstellen und führen Sie einen Auftrag aus, der CSV-Daten aus dem Amazon-S3-Quell-Bucket extrahiert, sie durch Entfernen und Umbenennen von Feldern transformiert und die JSON-formatierte Ausgabe in einen anderen Amazon-S3-Bucket lädt.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
```

```
        process.env.BUCKET_NAME,
        process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
};

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput> } getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
    const waitTimeInSeconds = 30;
    const { JobRun } = await getJobRun(jobName, jobRunId);

    if (!JobRun) {
        throw new Error(`Job run with id ${jobRunId} not found.`);
    }

    switch (JobRun.JobRunState) {
        case "FAILED":
        case "TIMEOUT":
        case "STOPPED":
            throw new Error(
                `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
            );
        case "RUNNING":
            break;
        case "SUCCEEDED":
            return;
        default:
            throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
    }

    log(
        `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
    );
    await wait(waitTimeInSeconds);
    return waitForJobRun(getJobRun, jobName, jobRunId);
};
```

```
/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
view the output.`
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
};
```

Listet Informationen über Auftragsausführungen auf und zeigt einige der transformierten Daten an.

```
const getJobRuns = (jobName) => {
```

```
const client = new GlueClient({});
const command = new GetJobRunsCommand({
  JobName: jobName,
});

return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput>}
getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
```

```
*
* @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
*/
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

Löscht alle Ressourcen, die von der Demo erstellt wurden.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });
};
```

```
    return client.send(command);
  };

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

/**
 *
 * @param {import('.././../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
  }
};
```



```
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('.././././actions/list-jobs.js').listJobs,
 *   deleteJob: import('.././././actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('.././././actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *   getTables: import('.././././actions/get-tables.js').getTables,
 *   deleteTable: import('.././././actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
```

```
/**
 * @param {{ prompter: { prompt: () => Promise<any>}}} context
 */
async (context) => {
  const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
    () => ({ TableList: null }),
  );

  if (TableList && TableList.length > 0) {
    /**
     * @type {{ tableNames: string[] }}
     */
    const { tableNames } = await context.prompter.prompt({
      name: "tableNames",
      type: "checkbox",
      message: "Let's clean up tables. Select tables to delete.",
      choices: TableList.map((t) => t.Name),
    });

    if (tableNames.length === 0) {
      log("No tables selected.");
    } else {
      log("Deleting tables.");
      await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
      log("Tables deleted.", { type: "success" });
    }
  }

  return { ...context };
};

/**
 * @param {import('.././././actions/delete-database.js').deleteDatabase}
deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('.././././actions/get-databases.js').getDatabases
```

```
* deleteDatabase: import('.././.././actions/delete-database.js').deleteDatabase
* }} config
*/
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };

const cleanUpCrawlerStep = async (context) => {
  log(`Deleting crawler.`);

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log(`Crawler is already deleted.`);
    } else {
      throw err;
    }
  }
}
```

```
    }  
  
    return { ...context };  
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

HealthImaging Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden HealthImaging.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo HealthImaging

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung beginnen HealthImaging.

SDK für JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- Einzelheiten zur API finden Sie [ListDatastores](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

CopyImageSet

Das folgende Codebeispiel zeigt, wie man es benutztCopyImageSet.

SDK für JavaScript (v3)

Hilfsfunktion zum Kopieren eines Bilddatensatzes.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 * set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 * image set.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = ""
) => {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  const response = await medicalImagingClient.send(
```

```

    new CopyImageSetCommand(params)
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxx',
  //   destinationImageSetProperties: {
  //     createdAt: 2023-09-27T19:46:21.824Z,
  //     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'COPYING',
  //     latestVersionId: '1',
  //     updatedAt: 2023-09-27T19:46:21.824Z
  //   },
  //   sourceImageSetProperties: {
  //     createdAt: 2023-09-22T14:49:26.427Z,
  //     imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxx:datastore/xxxxxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxx',
  //     imageSetId: 'xxxxxxxxxxxxxxxx',
  //     imageSetState: 'LOCKED',
  //     imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
  //     latestVersionId: '4',
  //     updatedAt: 2023-09-27T19:46:21.824Z
  //   }
  // }
  return response;
};

```

Kopiert einen Bilddatensatz ohne Ziel.

```

try {
  await copyImageSet(
    "12345678901234567890123456789012",

```

```
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.error(err);
}
```

Kopiert einen Bilddatensatz mit einem Ziel.

```
try {
  await copyImageSet(
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "4",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.error(err);
}
```

- Einzelheiten zur API finden Sie [CopyImageSet](#) unter AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

CreateDatastore

Das folgende Codebeispiel zeigt, wie man es benutzt CreateDatastore.

SDK für JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
```



```

*/
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};

```

- Einzelheiten zur API finden Sie [CreateDatastore](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

DeleteDatastore

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteDatastore`.

SDK für JavaScript (v3)

```

import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {

```

```

const response = await medicalImagingClient.send(
  new DeleteDatastoreCommand({ datastoreId })
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   datastoreStatus: 'DELETING'
// }

return response;
};

```

- Einzelheiten zur API finden Sie [DeleteDatastore](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

DeleteImageSet

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteImageSet`.

SDK für JavaScript (v3)

```

import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (

```

```
datastoreId = "xxxxxxxxxxxxxxxxxxxx",
imageSetId = "xxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eea5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie [DeleteImageSet](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

GetDICOMImportJob

Das folgende Codebeispiel zeigt, wie man es benutzt `GetDICOMImportJob`.

SDK für JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
```

```

import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxxxx"
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobProperties: {
  //     dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/dicom_import',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //     jobStatus: 'COMPLETED',
  //     outputS3Uri: 's3://health-imaging-dest/
  output_ct/xxxxxxxxxxxxxxxxxxxxxxxx-DicomImport-xxxxxxxxxxxxxxxxxxxxxxxxx/',
  //     submittedAt: 2023-09-19T17:27:25.143Z
  //   }
  // }

  return response;
};

```

- Einzelheiten zur API finden Sie unter [GetDICOM ImportJob](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

GetDatastore

Das folgende Codebeispiel zeigt, wie man es benutzt `GetDatastore`.

SDK für JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreProperties: {
  //     createdAt: 2023-08-04T18:50:36.239Z,
  //     datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //     datastoreName: 'my_datastore',
  //     datastoreStatus: 'ACTIVE',
  //     updatedAt: 2023-08-04T18:50:36.239Z
  //   }
  // }
  return response["datastoreProperties"];
}
```

```
};
```

- Einzelheiten zur API finden Sie [GetDatastore](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

GetImageFrame

Das folgende Codebeispiel zeigt, wie man es benutzt `GetImageFrame`.

SDK für JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 * image frame.
 * @param {string} datastoreId - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
  imageFrameFileName = "image.jph",
  datastoreId = "DATASTORE_ID",
  imageSetID = "IMAGE_SET_ID",
  imageFrameID = "IMAGE_FRAME_ID"
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    })
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);
};
```

```
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   contentType: 'application/octet-stream',
//   imageFrameBlob: <ref *1> IncomingMessage {}
// }
return response;
};
```

- Einzelheiten zur API finden Sie [GetImageFrame](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

GetImageSet

Das folgende Codebeispiel zeigt, wie man es benutzt `GetImageSet`.

SDK für JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxx",
```


GetImageSetMetadata

Das folgende Codebeispiel zeigt, wie man es benutzt `GetImageSetMetadata`.

SDK für JavaScript (v3)

Utility-Funktion zum Abrufen von Bilddatensatz-Metadaten.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  versionID = ""
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }

  const response = await medicalImagingClient.send(
    new GetImageSetMetadataCommand(params)
  );
  const buffer = await response.imageSetMetadataBlob.transformToByteArray();
  writeFileSync(metadataFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '5219b274-30ff-4986-8cab-48753de3a599',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//      totalRetryDelay: 0
// },
//      contentType: 'application/json',
//      contentEncoding: 'gzip',
//      imageSetMetadataBlob: <ref *1> IncomingMessage {}
// }

return response;
};
```

Ruft Bildsatz-Metadaten ohne Version ab.

```
try {
  await getImageSetMetadata(
    "metadata.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012"
  );
} catch (err) {
  console.log("Error", err);
}
```

Holen Sie sich Bildsatz-Metadaten mit Version.

```
try {
  await getImageSetMetadata(
    "metadata2.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1"
  );
} catch (err) {
  console.log("Error", err);
}
```

- Einzelheiten zur API finden Sie [GetImageSetMetadata](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

ListDICOMImportJobs

Das folgende Codebeispiel zeigt, wie man es benutzt `ListDICOMImportJobs`.

SDK für JavaScript (v3)

```
import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

  let jobSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    jobSummaries.push(...page["jobSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//      totalRetryDelay: 0
// },
//      jobSummaries: [
//          {
//              dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
//              datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//              endedAt: 2023-09-22T14:49:51.351Z,
//              jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//              jobName: 'test-1',
//              jobStatus: 'COMPLETED',
//              submittedAt: 2023-09-22T14:48:45.767Z
//          }
//      ]

return jobSummaries;
};
```

- Einzelheiten zur API finden Sie unter [ListDicom ImportJobs](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

ListDatastores

Das folgende Codebeispiel zeigt, wie man es benutzt `ListDatastores`.

SDK für JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };
};
```

```

const commandParams = {};
const paginator = paginateListDatastores(paginatorConfig, commandParams);

/**
 * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
 */
const datastoreSummaries = [];
for await (const page of paginator) {
  // Each page contains a list of `jobSummaries`. The list is truncated if is
  // larger than `pageSize`.
  datastoreSummaries.push(...page["datastoreSummaries"]);
  console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   datastoreSummaries: [
//     {
//       createdAt: 2023-08-04T18:49:54.429Z,
//       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       datastoreName: 'my_datastore',
//       datastoreStatus: 'ACTIVE',
//       updatedAt: 2023-08-04T18:49:54.429Z
//     }
//     ...
//   ]
// }

return datastoreSummaries;
};

```

- Einzelheiten zur API finden Sie [ListDatastores](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

ListImageSetVersions

Das folgende Codebeispiel zeigt, wie man es benutzt `ListImageSetVersions`.

SDK für JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx"
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
  const paginator = paginateListImageSetVersions(
    paginatorConfig,
    commandParams
  );

  let imageSetPropertiesList = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    larger than `pageSize`.
    imageSetPropertiesList.push(...page["imageSetPropertiesList"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
```

```
//      statusCode: 200,  
//      requestId: '74590b37-a002-4827-83f2-3c590279c742',  
//      extendedRequestId: undefined,  
//      cfId: undefined,  
//      attempts: 1,  
//      totalRetryDelay: 0  
//    },  
//    imageSetPropertiesList: [  
//      {  
//        ImageSetWorkflowStatus: 'CREATED',  
//        createdAt: 2023-09-22T14:49:26.427Z,  
//        imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxx',  
//        imageSetState: 'ACTIVE',  
//        versionId: '1'  
//      }  
//    ]  
//  }  
return imageSetPropertiesList;  
};
```

- Einzelheiten zur API finden Sie [ListImageSetVersions](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

ListTagsForResource

Das folgende Codebeispiel zeigt, wie man es benutzt `ListTagsForResource`.

SDK für JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";  
import { medicalImagingClient } from "../libs/medicalImagingClient.js";  
  
/**  
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store  
 * or image set.  
 */
```

```
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [ListTagsForResource](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

SearchImageSets

Das folgende Codebeispiel zeigt, wie man es benutzt `SearchImageSets`.

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bilddatensätzen.

```
import {paginateSearchImageSets} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";
```



```
/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
 search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
 criteria sort.
 */
export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  searchCriteria = {}
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: searchCriteria,
  };

  const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

  const imageSetsMetadataSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
larger than `pageSize`.
    imageSetsMetadataSummaries.push(...page["imageSetsMetadataSummaries"]);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetsMetadataSummaries: [
  //     {
  //       DICOMTags: [Object],
  //       createdAt: "2023-09-19T16:59:40.551Z",
  //       imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',

```

```
//             updatedAt: "2023-09-19T16:59:40.551Z",
//             version: 1
//         }]}
//     }

    return imageSetsMetadataSummaries;
};
```

Anwendungsfall #1: EQUAL-Operator.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [{DICOMPatientId: "1234567"}],
                operator: "EQUAL",
            },
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

Anwendungsfall #2: BETWEEN-Operator, der DICOM StudyDate und StudyTime DICOM verwendet.

```
const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {
                        DICOMStudyDateAndTime: {
                            DICOMStudyDate: "19900101",
                            DICOMStudyTime: "000000",
                        }
                    }
                ]
            }
        ]
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}
```

```

        },
        },
        {
            DICOMStudyDateAndTime: {
                DICOMStudyDate: "20230901",
                DICOMStudyTime: "000000",
            },
        },
    ],
    operator: "BETWEEN",
},
]
};

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}

```

Anwendungsfall #3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```

const datastoreId = "12345678901234567890123456789012";

try {
    const searchCriteria = {
        filters: [
            {
                values: [
                    {createdAt: new Date("1985-04-12T23:20:50.52Z")},
                    {createdAt: new Date()},
                ],
                operator: "BETWEEN",
            },
        ],
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}

```

Anwendungsfall #4: EQUAL-Operator für DICOM SeriesInstance UID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge im updatedAt-Feld.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          {updatedAt: new Date("1985-04-12T23:20:50.52Z")},
          {updatedAt: new Date()},
        ],
        operator: "BETWEEN",
      },
      {
        values: [
          {DICOMSeriesInstanceUID:
"1.1.123.123456.1.12.1.1234567890.1234.12345678.123"},
        ],
        operator: "EQUAL",
      },
    ],
    sort: {
      sortOrder: "ASC",
      sortField: "updatedAt",
    }
  };

  await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}
```

- Einzelheiten zur API finden Sie in der API-Referenz. [SearchImageSets](#) AWS SDK for JavaScript

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

StartDICOMImportJob

Das folgende Codebeispiel zeigt die Verwendung `StartDICOMImportJob`.

SDK für JavaScript (v3)

```
import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
stored.
 */
export const startDicomImportJob = async (
  jobName = "test-1",
  datastoreId = "12345678901234567890123456789012",
  dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
  inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",
  outputS3Uri = "s3://medical-imaging-output/job_output/"
) => {
  const response = await medicalImagingClient.send(
    new StartDICOMImportJobCommand({
      jobName: jobName,
      datastoreId: datastoreId,
      dataAccessRoleArn: dataAccessRoleArn,
      inputS3Uri: inputS3Uri,
      outputS3Uri: outputS3Uri,
    })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
//     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//     jobStatus: 'SUBMITTED',
//     submittedAt: 2023-09-22T14:48:45.767Z
// }
return response;
};
```

- Einzelheiten zur API finden Sie unter [StartDicom ImportJob](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

TagResource

Das folgende Codebeispiel zeigt die Verwendung `TagResource`.

SDK für JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
  xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
};
```

```

// {
//   '$metadata': {
//     httpStatusCode: 204,
//     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }

return response;
};

```

- Einzelheiten zur API finden Sie [TagResource](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

UntagResource

Das folgende Codebeispiel zeigt die Verwendung `UntagResource`.

SDK für JavaScript (v3)

```

import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(

```

```
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [UntagResource](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

UpdateImageSetMetadata

Das folgende Codebeispiel zeigt die Verwendung `UpdateImageSetMetadata`.

SDK für JavaScript (v3)

```
import {UpdateImageSetMetadataCommand} from "@aws-sdk/client-medical-imaging";
import {medicalImagingClient} from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 */
export const updateImageSetMetadata = async (datastoreId = "xxxxxxxxxx",
                                             imageSetId = "xxxxxxxxxx",
```



```

        latestVersionId = "1",
        updateMetadata = '{}') => {
const response = await medicalImagingClient.send(
    new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata
    })
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   createdAt: 2023-09-22T14:49:26.427Z,
//   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//   imageSetState: 'LOCKED',
//   imageSetWorkflowStatus: 'UPDATING',
//   latestVersionId: '4',
//   updatedAt: 2023-09-27T19:41:43.494Z
// }
return response;
};

```

Anwendungsfall #1: Fügen Sie ein Attribut ein oder aktualisieren Sie es.

```

const insertAttributes =
    JSON.stringify({
        "SchemaVersion": 1.1,
        "Study": {
            "DICOM": {
                "StudyDescription": "CT CHEST"
            }
        }
    });

```

```
const updateMetadata = {
  "DICOMUpdates": {
    "updateableAttributes":
      new TextEncoder().encode(insertAttributes)
  }
};

await updateImageSetMetadata(datastoreId, imageSetID,
  versionID, updateMetadata);
```

Anwendungsfall #2: Entfernen Sie ein Attribut.

```
// Attribute key and value must match the existing attribute.
const remove_attribute =
  JSON.stringify({
    "SchemaVersion": 1.1,
    "Study": {
      "DICOM": {
        "StudyDescription": "CT CHEST"
      }
    }
  });

const updateMetadata = {
  "DICOMUpdates": {
    "removableAttributes":
      new TextEncoder().encode(remove_attribute)
  }
};

await updateImageSetMetadata(datastoreId, imageSetID,
  versionID, updateMetadata);
```

Anwendungsfall #3: Eine Instanz entfernen.

```
const remove_instance =
  JSON.stringify({
    "SchemaVersion": 1.1,
    "Study": {
      "Series": {
```

```
        "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {
            "Instances": {
                "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {}
            }
        }
    });

    const updateMetadata = {
        "DICOMUpdates": {
            "removableAttributes":
                new TextEncoder().encode(remove_instance)
        }
    };

    await updateImageSetMetadata(datastoreId, imageSetID,
        versionID, updateMetadata);
```

- Einzelheiten zur API finden Sie [UpdateImageSetMetadata](#) in der AWS SDK for JavaScript API-Referenz.

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Szenarien

Beginnen Sie mit Bildsets und Bildrahmen

Das folgende Codebeispiel zeigt, wie Sie DICOM-Dateien importieren und Bildrahmen herunterladen.
HealthImaging

Die Implementierung ist als Workflow-Befehlszeilenanwendung strukturiert.

- Richten Sie Ressourcen für einen DICOM-Import ein.
- Importieren Sie DICOM-Dateien in einen Datenspeicher.
- Rufen Sie die Bilddatensatz-IDs für den Importauftrag ab.

- Rufen Sie die Bildrahmen-IDs für die Bilddatensätze ab.
- Laden Sie die Bildrahmen herunter, dekodieren Sie sie und überprüfen Sie sie.
- Ressourcen bereinigen.

SDK für JavaScript (v3)

index.js- Schritte orchestrieren.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "./deploy-steps.js";
import {
  doCopy,
  selectDataset,
  copyDataset,
  outputCopiedObjects,
} from "./dataset-steps.js";
import {
  doImport,
  outputImportJobStatus,
  startDICOMImport,
  waitForImportJobCompletion,
} from "./import-steps.js";
import {
```

```
    getManifestFile,
    outputImageSetIds,
    parseManifestFile,
  } from "./image-set-steps.js";
import {
  getImageSetMetadata,
  outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
  confirmCleanup,
  deleteImageSets,
  deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
  deploy: new Scenario(
    "Deploy Resources",
    [
      deployStack,
      getStackName,
      getDatastoreName,
      getAccountId,
      createStack,
      waitForStackCreation,
      outputState,
      saveState,
    ],
    context,
  ),
  demo: new Scenario(
    "Run Demo",
    [
      loadState,
      doCopy,
      selectDataset,
      copyDataset,
      outputCopiedObjects,
      doImport,
      startDICOMImport,
      waitForImportJobCompletion,
      outputImportJobStatus,
    ],
  ),
};
```

```
    getManifestFile,  
    parseManifestFile,  
    outputImageSetIds,  
    getImageSetMetadata,  
    outputImageFrameIds,  
    doVerify,  
    decodeAndVerifyImages,  
    saveState,  
  ],  
  context,  
),  
destroy: new Scenario(  
  "Clean Up Resources",  
  [loadState, confirmCleanup, deleteImageSets, deleteStack],  
  context,  
),  
};  
  
// Call function if run directly  
import { fileURLToPath } from "url";  
if (process.argv[1] === fileURLToPath(import.meta.url)) {  
  parseScenarioArgs(scenarios);  
}
```

deploy-steps.js- Ressourcen bereitstellen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import fs from "node:fs/promises";  
import path from "node:path";  
  
import {  
  CloudFormationClient,  
  CreateStackCommand,  
  DescribeStacksCommand,  
} from "@aws-sdk/client-cloudformation";  
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";  
  
import {  
  ScenarioAction,  
  ScenarioInput,  
  ScenarioOutput,  
}
```

```
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../workflows/healthimaging_image_sets/resources/cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
```

```
async (/** @type {} */ state) => {
  const stackName = state.getStackName;
  const dataStoreName = state.getDataStoreName;
  const accountId = state.accountId;

  const command = new CreateStackCommand({
    StackName: stackName,
    TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
    Capabilities: ["CAPABILITY_IAM"],
    Parameters: [
      {
        ParameterKey: "datastoreName",
        ParameterValue: dataStoreName,
      },
      {
        ParameterKey: "userAccountID",
        ParameterValue: accountId,
      },
    ],
  });

  const response = await cfnClient.send(command);
  state.stackId = response.StackId;
},
{ skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName == state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
```



```

        acc[output.OutputKey] = output.OutputValue;
        return acc;
    }, {}));
} else {
    throw new Error(
        `Stack creation failed with status: ${stack.StackStatus}`,
    );
}
});
},
{
    skipWhen: (/** @type {} */ state) => !state.deployStack,
},
);

export const outputState = new ScenarioOutput(
    "outputState",
    (/** @type {} */ state) => {
        /**
         * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
         string }}}
         */
        const { stackOutputs } = state;
        return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}
`;
    },
    { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

```

dataset-steps.js- Kopieren Sie DICOM-Dateien.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    S3Client,
    CopyObjectCommand,
    ListObjectsV2Command,
} from "@aws-sdk/client-s3";

```

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   doCopy: boolean
 * }}} State
 */

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
```

```
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = `input/`;
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);

    const copyPromises = objects.Contents.map((object) => {
      const sourceKey = object.Key;
      const destinationKey = `${inputPrefix}${sourceKey}
        .split("/")
        .slice(1)
        .join("/")}`;

      const copyCommand = new CopyObjectCommand({
        Bucket: inputBucket,
        CopySource: `/${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });

      return s3Client.send(copyCommand);
    });
  }
);
```

```
});

    const results = await Promise.all(copyPromises);
    state.copiedObjects = results.length;
  },
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);
```

import-steps.js- Startet den Import in den Datenspeicher.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
```

```
    },
  );

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreId,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({
      datastoreId: state.stackOutputs.DatastoreId,
      jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await medicalImagingClient.send(command);
      const jobStatus = response.jobProperties?.jobStatus;
      if (!jobStatus || jobStatus === "IN_PROGRESS") {
        throw new Error("Import job is still in progress");
      }
      if (jobStatus === "COMPLETED") {
        state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
      } else {
        throw new Error(`Import job failed with status: ${jobStatus}`);
      }
    });
  }
);
```

```

    }
  });
},
);

export const outputImportJobStatus = new ScenarioOutput(
  "outputImportJobStatus",
  (state) =>
    `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);

```

`image-set-steps.js`- Holen Sie sich die IDs von Bilddatensätzen.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],
 * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
 * }} State
 */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (/** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;

```

```

    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce(
        (imageSetIds, next) => {
          return { ...imageSetIds, [next.imageSetId]: next.imageSetId };
        },
        {},
      );
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}` ,
);

```

image-frame-steps.js- Holen Sie sich Bildrahmen-IDs.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,

```

```
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "zlib";
import { promisify } from "util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */
```



```
*/

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
        imageSetId,
      });

      const response = await medicalImagingClient.send(command);
      const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
      const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
      const imageSetMetadata = JSON.parse(decompressedMetadata.toString());
    }
  }
);
```

```

        outputMetadata.push(imageSetMetadata);
    }

    state.imageSetMetadata = outputMetadata;
  },
);

export const outputImageFrameIds = new ScenarioOutput(
  "outputImageFrameIds",
  (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
    let output = "";

    for (const metadata of state.imageSetMetadata) {
      const imageSetId = metadata.ImageSetID;
      /** @type {DICOMMetadata[]} */
      const instances = Object.values(metadata.Study.Series).flatMap(
        (series) => {
          return Object.values(series.Instances);
        },
      );
      const imageFrameIds = instances.flatMap((instance) =>
        instance.ImageFrames.map((frame) => frame.ID),
      );

      output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
  "\n",
)}\n\n`;
    }

    return output;
  },
  { slow: false },
);

```

verify-steps.js- Überprüfen Sie die Bildrahmen. Die Bibliothek [zur Überprüfung von AWS HealthImaging Pixeldaten](#) wurde zur Überprüfung verwendet.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { spawn } from "node:child_process";

```

```
import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */
```

```
/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;
      const imageSetId = metadata.ImageSetID;

      for (const [seriesInstanceId, series] of Object.entries(
        metadata.Study.Series,
      )) {
        for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
          console.log(
```

```

        `Verifying image set ${imageSetId} with series ${seriesInstanceId} and
sop ${sopInstanceId}` ,
    );
    const child = spawn(
        "node",
        [
            verificationTool,
            datastoreId,
            imageSetId,
            seriesInstanceId,
            sopInstanceId,
        ],
        { stdio: "inherit" },
    );

    await new Promise((resolve, reject) => {
        child.on("exit", (code) => {
            if (code === 0) {
                resolve();
            } else {
                reject(
                    new Error(
                        `Verification tool exited with code ${code} for image set
${imageSetId}`,
                    ),
                );
            }
        });
    });
}
}
},
);

```

clean-up-steps.js- Zerstöre Ressourcen.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
    CloudFormationClient,
    DeleteStackCommand,

```

```
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
```

```
*/

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (/** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });
    }
  });
```

```

    try {
      await medicalImagingClient.send(command);
      console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
    } catch (e) {
      if (e instanceof Error) {
        if (e.name === "ConflictException") {
          console.log(`Image set ${metadata.ImageSetID} already deleted`);
        }
      }
    }
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {{}} */ state) => !state.confirmCleanup,
  },
);

```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [DeleteImageSet](#)
 - [Holen Sie sich DICOM ImportJob](#)
 - [GetImageFrame](#)
 - [GetImageSetMetadata](#)

- [SearchImageSets](#)
- [Starten Sie Dicom ImportJob](#)

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Einen Datenspeicher taggen

Das folgende Codebeispiel zeigt, wie ein HealthImaging Datenspeicher markiert wird.

SDK für JavaScript (v3)

Um einen Datenspeicher zu taggen.

```
try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const tags = {
    Deployment: "Development",
  };
  await tagResource(datastoreArn, tags);
} catch (e) {
  console.log(e);
}
```

Die Hilfsfunktion zum Markieren einer Ressource.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *
   - For example: {"Deployment" : "Development"}
 */
```

```

export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};

```

Um Tags für einen Datenspeicher aufzulisten.

```

try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const { tags } = await listTagsForResource(datastoreArn);
  console.log(tags);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Auflisten der Tags einer Ressource.

```

import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**

```

```

* @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
*/
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};

```

Um die Markierung eines Datenspeichers aufzuheben.

```

try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(datastoreArn, keys);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Entkennzeichnen einer Ressource.

```

import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

```

```
/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Einen Bilddatensatz taggen

Das folgende Codebeispiel zeigt, wie Sie einen HealthImaging Bildsatz taggen.

SDK für JavaScript (v3)

Um einen Bilddatensatz zu taggen.

```
try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const tags = {
    Deployment: "Development",
  };
  await tagResource(imagesetArn, tags);
} catch (e) {
  console.log(e);
}
```

Die Hilfsfunktion zum Markieren einer Ressource.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 *
   - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {}
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags })
  );
  console.log(response);
  // {
```

```
//     '$metadata': {
//       httpStatusCode: 204,
//       requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
//       extendedRequestId: undefined,
//       cfId: undefined,
//       attempts: 1,
//       totalRetryDelay: 0
//     }
// }

return response;
};
```

Um Tags für einen Bilddatensatz aufzulisten.

```
try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const { tags } = await listTagsForResource(imagesetArn);
  console.log(tags);
} catch (e) {
  console.log(e);
}
```

Die Hilfsfunktion zum Auflisten der Tags einer Ressource.

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi"
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn })
  );
};
```

```

console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   tags: { Deployment: 'Development' }
// }

return response;
};

```

Um die Markierung eines Bilddatensatzes aufzuheben.

```

try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(imagesetArn, keys);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Entkennzeichnen einer Ressource.

```

import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (

```

```
resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
tagKeys = []
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys })
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

Note

Es gibt noch mehr dazu. GitHub Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

IAM-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit IAM Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo IAM

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von IAM beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
   * async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
```

```
for await (const page of paginator) {
  if (page.Policies) {
    page.Policies.forEach((p) => {
      console.log(`${p.PolicyName}`);
      policyCount++;
    });
  }
}
console.log(`Found ${policyCount} policies.`);
};
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AttachRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `AttachRolePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Anfügen einer Richtlinie.

```
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);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [AttachRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
```

```
myRolePolicies.forEach(function (val, index, array) {
  if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
    console.log(
      "AmazonDynamoDBFullAccess is already attached to this role."
    );
    process.exit();
  }
});
var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
  RoleName: process.argv[2],
};
iam.attachRolePolicy(params, function (err, data) {
  if (err) {
    console.log("Unable to attach policy to role", err);
  } else {
    console.log("Role attached successfully");
  }
});
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [AttachRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

CreateAccessKey

Das folgende Codebeispiel zeigt die Verwendung `CreateAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Zugriffsschlüssel.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [CreateAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

CreateAccountAlias

Das folgende Codebeispiel zeigt die Verwendung `CreateAccountAlias`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Konto-Alias.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateAccountAlias](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateAccountAlias](#) in der AWS SDK for JavaScript API-Referenz.

CreateGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [CreateGroup](#) in der AWS SDK for JavaScript API-Referenz.

CreateInstanceProfile

Das folgende Codebeispiel zeigt die Verwendung `CreateInstanceProfile`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```


- Einzelheiten zur API finden Sie [CreateInstanceProfile](#) in der AWS SDK for JavaScript API-Referenz.

CreatePolicy

Das folgende Codebeispiel zeigt die Verwendung `CreatePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie die Richtlinie.

```
import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Action: "*",
          Resource: "*",
        },
      ],
    }),
    PolicyName: policyName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreatePolicy](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};
```

```
var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreatePolicy](#) in der AWS SDK for JavaScript API-Referenz.

CreateRole

Das folgende Codebeispiel zeigt die Verwendung `CreateRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie die -Rolle.

```
import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
  const command = new CreateRoleCommand({
```

```
AssumeRolePolicyDocument: JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Principal: {
        Service: "lambda.amazonaws.com",
      },
      Action: "sts:AssumeRole",
    },
  ],
}),
RoleName: roleName,
});

return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateRole](#) in der AWS SDK for JavaScript API-Referenz.

CreateSAMLProvider

Das folgende Codebeispiel zeigt die Verwendung `CreateSAMLProvider`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import * as path from "path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.
```

```
* For more information on generating this document,
see https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_providers_create_saml.html#samlstep1.
*/
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere API-Informationen finden Sie unter [CreateSAMLProvider](#) in der API-Referenz für AWS SDK for JavaScript .

CreateServiceLinkedRole

Das folgende Codebeispiel zeigt die Verwendung `CreateServiceLinkedRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen einer serviceverknüpften Rolle.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
    // that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
    // latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
account",
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    } else {
      throw caught;
    }
  }
}
```

```
}  
};
```

- Einzelheiten zur API finden Sie [CreateServiceLinkedRole](#) in der AWS SDK for JavaScript API-Referenz.

CreateUser

Das folgende Codebeispiel zeigt die Verwendung `CreateUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Benutzer.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} name  
 */  
export const createUser = (name) => {  
  const command = new CreateUserCommand({ UserName: name });  
  return client.send(command);  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateUser](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  Username: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateUser](#) in der AWS SDK for JavaScript API-Referenz.

DeleteAccessKey

Das folgende Codebeispiel zeigt die Verwendung `DeleteAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie den Zugriffsschlüssel.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};


iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

DeleteAccountAlias

Das folgende Codebeispiel zeigt die Verwendung `DeleteAccountAlias`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie den Konto-Alias.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";


const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccountAlias](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccountAlias](#) in der AWS SDK for JavaScript API-Referenz.

DeleteGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
```

```
});  
  
const response = await client.send(command);  
console.log(response);  
return response;  
};
```

- Einzelheiten zur API finden Sie [DeleteGroup](#) in der AWS SDK for JavaScript API-Referenz.

DeleteInstanceProfile

Das folgende Codebeispiel zeigt die Verwendung `DeleteInstanceProfile`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const client = new IAMClient({});  
await client.send(  
  new DeleteInstanceProfileCommand({  
    InstanceProfileName: NAMES.instanceProfileName,  
  }),  
);
```

- Einzelheiten zur API finden Sie [DeleteInstanceProfile](#) in der AWS SDK for JavaScript API-Referenz.

DeletePolicy

Das folgende Codebeispiel zeigt die Verwendung `DeletePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie die Richtlinie.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeletePolicy](#) in der AWS SDK for JavaScript API-Referenz.

DeleteRole

Das folgende Codebeispiel zeigt die Verwendung `DeleteRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie die Rolle.

```
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);
};
```

- Einzelheiten zur API finden Sie [DeleteRole](#) in der AWS SDK for JavaScript API-Referenz.

DeleteRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `DeleteRolePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
};
```

```
    return client.send(command);
  };
```

- Einzelheiten zur API finden Sie [DeleteRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

DeleteSAMLProvider

Das folgende Codebeispiel zeigt die Verwendung `DeleteSAMLProvider`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
  const command = new DeleteSAMLProviderCommand({
    SAMLProviderArn: providerArn,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere API-Informationen finden Sie unter [DeleteSAMLProvider](#) in der API-Referenz für AWS SDK for JavaScript .

DeleteServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `DeleteServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie ein Serverzertifikat.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

DeleteServiceLinkedRole

Das folgende Codebeispiel zeigt die Verwendung `DeleteServiceLinkedRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteServiceLinkedRole](#) in der AWS SDK for JavaScript API-Referenz.

DeleteUser

Das folgende Codebeispiel zeigt die Verwendung `DeleteUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie den Benutzer.


```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteUser](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteUser](#) in der AWS SDK for JavaScript API-Referenz.

DetachRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `DetachRolePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Trennen Sie die Richtlinie.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";


const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DetachRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

```
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DetachRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

GetAccessKeyLastUsed

Das folgende Codebeispiel zeigt die Verwendung `GetAccessKeyLastUsed`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie den Zugriffsschlüssel ab.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} accessKeyId
 */
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }
}
```

```
    }  
  
    return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetAccessKeyLastUsed](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create the IAM service object  
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });  
  
iam.getAccessKeyLastUsed(  
  { AccessKeyId: "ACCESS_KEY_ID" },  
  function (err, data) {  
    if (err) {  
      console.log("Error", err);  
    } else {  
      console.log("Success", data.AccessKeyLastUsed);  
    }  
  }  
);
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetAccessKeyLastUsed](#) in der AWS SDK for JavaScript API-Referenz.

GetAccountPasswordPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetAccountPasswordPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die Passwort-Richtlinie des Kontos ab.

```
import {
  GetAccountPasswordPolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const getAccountPasswordPolicy = async () => {
  const command = new GetAccountPasswordPolicyCommand({});

  const response = await client.send(command);
  console.log(response.PasswordPolicy);
  return response;
};
```

- Einzelheiten zur API finden Sie [GetAccountPasswordPolicy](#) in der AWS SDK for JavaScript API-Referenz.

GetPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetPolicy`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die Richtlinie ab.

```
import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";


const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {
  const command = new GetPolicyCommand({
    PolicyArn: policyArn,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetPolicy](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetPolicy](#) in der AWS SDK for JavaScript API-Referenz.

GetRole

Das folgende Codebeispiel zeigt die Verwendung `GetRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die Rolle ab.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
*
* @param {string} roleName
*/
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetRole](#) in der AWS SDK for JavaScript API-Referenz.

GetServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `GetServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie ein Serverzertifikat ab.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });
};
```

```
const response = await client.send(command);
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

GetServiceLinkedRoleDeletionStatus

Das folgende Codebeispiel zeigt die Verwendung `GetServiceLinkedRoleDeletionStatus`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetServiceLinkedRoleDeletionStatus](#) in der AWS SDK for JavaScript API-Referenz.

ListAccessKeys

Das folgende Codebeispiel zeigt die Verwendung `ListAccessKeys`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Zugriffsschlüssel auf.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
      );
    }
  }
}
```

```
    );  
  } else {  
    break;  
  }  
}  
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListAccessKeys](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create the IAM service object  
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });  
  
var params = {  
  MaxItems: 5,  
  UserName: "IAM_USER_NAME",  
};  
  
iam.listAccessKeys(params, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data);  
  }  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [ListAccessKeys](#) in der AWS SDK for JavaScript API-Referenz.

ListAccountAliases

Das folgende Codebeispiel zeigt die Verwendung `ListAccountAliases`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Konto-Aliase auf.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 */
export async function* listAccountAliases() {
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });

  let response = await client.send(command);

  while (response.AccountAliases?.length) {
    for (const alias of response.AccountAliases) {
      yield alias;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccountAliasesCommand({
          Marker: response.Marker,
          MaxItems: 5,
        })
      );
    }
  }
}
```

```
    );  
  } else {  
    break;  
  }  
}  
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListAccountAliases](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create the IAM service object  
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });  
  
iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data);  
  }  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListAccountAliases](#) in der AWS SDK for JavaScript API-Referenz.

ListAttachedRolePolicies

Das folgende Codebeispiel zeigt die Verwendung `ListAttachedRolePolicies`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Richtlinien auf, die an eine Rolle angefügt sind.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
```

```
        RoleName: roleName,  
        Marker: response.Marker,  
    }},  
    );  
} else {  
    break;  
}  
}  
}
```

- Einzelheiten zur API finden Sie [ListAttachedRolePolicies](#) in der AWS SDK for JavaScript API-Referenz.

ListGroups

Das folgende Codebeispiel zeigt die Verwendung `ListGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Gruppen auf.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 * A generator function that handles paginated results.  
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.  
 */  
export async function* listGroups() {  
    const command = new ListGroupsCommand({  
        MaxItems: 10,  
    });
```

```
let response = await client.send(command);

while (response.Groups?.length) {
  for (const group of response.Groups) {
    yield group;
  }

  if (response.IsTruncated) {
    response = await client.send(
      new ListGroupsCommand({
        Marker: response.Marker,
        MaxItems: 10,
      }),
    );
  } else {
    break;
  }
}
```

- Einzelheiten zur API finden Sie [ListGroups](#) in der AWS SDK for JavaScript API-Referenz.

ListPolicies

Das folgende Codebeispiel zeigt die Verwendung `ListPolicies`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listet Sie die Richtlinien auf.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListPoliciesCommand({
          Marker: response.Marker,
          MaxItems: 10,
          OnlyAttached: false,
          Scope: "Local",
        })),
    );
  } else {
    break;
  }
}
}
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der AWS SDK for JavaScript API-Referenz.

ListRolePolicies

Das folgende Codebeispiel zeigt die Verwendung `ListRolePolicies`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listet Sie die Richtlinien auf.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 *
 * @param {string} roleName
 */
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        })
      );
    }
  }
}
```

```
    } else {  
      break;  
    }  
  }  
}
```

- Einzelheiten zur API finden Sie [ListRolePolicies](#) in der AWS SDK for JavaScript API-Referenz.

ListRoles

Das folgende Codebeispiel zeigt die Verwendung `ListRoles`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Rollen auf.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 * A generator function that handles paginated results.  
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.  
 *  
 */  
export async function* listRoles() {  
  const command = new ListRolesCommand({  
    MaxItems: 10,  
  });  
  
  /**  
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}  
   */
```



```
*/
let response = await client.send(command);

while (response?.Roles?.length) {
  for (const role of response.Roles) {
    yield role;
  }

  if (response.IsTruncated) {
    response = await client.send(
      new ListRolesCommand({
        Marker: response.Marker,
      }),
    );
  } else {
    break;
  }
}
}
```

- Einzelheiten zur API finden Sie [ListRoles](#) in der AWS SDK for JavaScript API-Referenz.

ListSAMLProviders

Das folgende Codebeispiel zeigt die Verwendung `ListSAMLProviders`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die SAML-Anbieter auf.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere API-Informationen finden Sie unter [ListSAMLProviders](#) in der API-Referenz für AWS SDK for JavaScript .

ListServerCertificates

Das folgende Codebeispiel zeigt die Verwendung `ListServerCertificates`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Zertifikate auf.

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify this.
 *
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);
```

```
while (response.ServerCertificateMetadataList?.length) {
  for await (const cert of response.ServerCertificateMetadataList) {
    yield cert;
  }

  if (response.IsTruncated) {
    response = await client.send(new ListServerCertificatesCommand({}));
  } else {
    break;
  }
}
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListServerCertificates](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListServerCertificates](#) in der AWS SDK for JavaScript API-Referenz.

ListUsers

Das folgende Codebeispiel zeigt die Verwendung `ListUsers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Benutzer auf.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);
  response.Users?.forEach(({ UserName, CreateDate }) => {
    console.log(`${UserName} created on: ${CreateDate}`);
  });
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListUsers](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};


iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
    });
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListUsers](#) in der AWS SDK for JavaScript API-Referenz.

PutRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `PutRolePolicy`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::some-test-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [PutRolePolicy](#) in der AWS SDK for JavaScript API-Referenz.

UpdateAccessKey

Das folgende Codebeispiel zeigt die Verwendung `UpdateAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Aktualisieren Sie den Zugriffsschlüssel.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};
```



```
iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateAccessKey](#) in der AWS SDK for JavaScript API-Referenz.

UpdateServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `UpdateServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Aktualisieren Sie ein Serverzertifikat.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
 */
export const updateServerCertificate = (currentName, newName) => {
  const command = new UpdateServerCertificateCommand({
    ServerCertificateName: currentName,
    NewServerCertificateName: newName,
  });
};
```

```
    return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

UpdateUser

Das folgende Codebeispiel zeigt die Verwendung `UpdateUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Aktualisieren Sie den Benutzer.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentUserName
 * @param {string} newUserName
 */
export const updateUser = (currentUserName, newUserName) => {
  const command = new UpdateUserCommand({
    UserName: currentUserName,
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateUser](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};


iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateUser](#) in der AWS SDK for JavaScript API-Referenz.

UploadServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `UploadServerCertificate`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }
  }

  throw err;
}
};
```

```
/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [UploadServerCertificate](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Gruppe von Amazon EC2 Auto Scaling, um Amazon Elastic Compute Cloud (Amazon EC2)-Instances basierend auf einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2-Instance einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 * - deploy
 * - demo
 * - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios);
}
```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
```



```
    CreatePolicyCommand,
    CreateRoleCommand,
    CreateInstanceProfileCommand,
    AddRoleToInstanceProfileCommand,
    AttachRolePolicyCommand,
    waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "./constants.js";
import { initParamsSteps } from "./steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
    new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
    new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
        type: "confirm",
    }),
    new ScenarioAction(
        "handleConfirmDeployment",
        (c) => c.confirmDeployment === false && process.exit(),
    ),
    new ScenarioOutput(
```

```
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      }),
    );
    await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
  }),
  new ScenarioOutput(
    "createdTable",
    MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioOutput(
    "populatingTable",
    MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),

```

```
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,
    }),
  );

  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
```

```
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  );
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});
  return client.send(
    new CreateRoleCommand({
      RoleName: NAMES.instanceRoleName,
      AssumeRolePolicyDocument: readFileSync(
        join(ROOT, "assume-role-policy.json"),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
```

```
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  );
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
```

```

    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  // snippet-start:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,

```

```
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  })),
  // snippet-end:[javascript.v3.wkflw.resilient.CreateLaunchTemplate]
);
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({
        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
          LaunchTemplateName: NAMES.launchTemplateName,
          Version: "$Default",
        },
      },
      {
        MinSize: 3,
      }
    )
  );
});
```

```

        MaxSize: 3,
      )),
    ),
  );
}),
new ScenarioOutput(
  "createdAutoScalingGroup",
  /**
   * @param {{ availabilityZoneNames: string[] }} state
   */
  (state) =>
    MESSAGES.createdAutoScalingGroup
      .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
      .replace(
        "${AVAILABILITY_ZONE_NAMES}",
        state.availabilityZoneNames.join(", "),
      ),
  ),
  ),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
  type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeVpcs]
  const client = new EC2Client({});
  const { Vpcs } = await client.send(
    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeVpcs]
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeSubnets]
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [

```



```

        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
    ],
    }),
);
// snippet-end:[javascript.v3.wkflw.resilient.DescribeSubnets]
state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
    "gotSubnets",
    /**
     * @param {{ subnets: string[] }} state
     */
    (state) =>
        MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
    "creatingLoadBalancerTargetGroup",
    MESSAGES.creatingLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
        new CreateTargetGroupCommand({
            Name: NAMES.loadBalancerTargetGroupName,
            Protocol: "HTTP",
            Port: 80,
            HealthCheckPath: "/healthcheck",
            HealthCheckIntervalSeconds: 10,
            HealthCheckTimeoutSeconds: 5,
            HealthyThresholdCount: 2,
            UnhealthyThresholdCount: 2,
            VpcId: state.defaultVpc,
        }),
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateTargetGroup]
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;

```

```
    state.targetGroupPort = targetGroup.Port;
  }),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
    // snippet-end:[javascript.v3.wkflw.resilient.CreateLoadBalancer]
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    // snippet-start:[javascript.v3.wkflw.resilient.CreateListener]
    const client = new ElasticLoadBalancingV2Client({});
```

```
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateListener]
const listener = Listeners[0];
state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.AttachTargetGroup]
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.AttachTargetGroup]
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
  "verifyInboundPort",
```

```
/**
 *
 * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
 */
async (state) => {
  const client = new EC2Client({});
  const { SecurityGroups } = await client.send(
    new DescribeSecurityGroupsCommand({
      Filters: [{ Name: "group-name", Values: ["default"] }],
    }),
  );
  if (!SecurityGroups) {
    state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
  }
  state.defaultSecurityGroup = SecurityGroups[0];

  /**
   * @type {string}
   */
  const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
  state.myIp = ipResponse.trim();
  const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
    ({ IpRanges }) =>
      IpRanges.some(
        ({ CidrIp }) =>
          CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
      ),
  )
  .filter(({ IpProtocol }) => IpProtocol === "tcp")
  .filter(({ FromPort }) => FromPort === 80);

  state.myIpRules = myIpRules;
},
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",

```

```
        JSON.stringify(state.myIpRules, null, 2),
    );
    } else {
        return MESSAGES.noIpRules;
    }
},
),
new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return false;
        } else {
            return MESSAGES.noIpRules;
        }
    },
    { type: "confirm" },
),
new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
        if (!state.shouldAddInboundRule) {
            return;
        }

        const client = new EC2Client({});
        await client.send(
            new AuthorizeSecurityGroupIngressCommand({
                GroupId: state.defaultSecurityGroup.GroupId,
                CidrIp: `${state.myIp}/32`,
                FromPort: 80,
                ToPort: 80,
                IpProtocol: "tcp",
            }),
        );
    },
),
),
```

```
new ScenarioOutput("addedInboundRule", (state) => {
  if (state.shouldAddInboundRule) {
    return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
  } else {
    return false;
  }
}),
new ScenarioOutput("verifyingEndpoint", (state) =>
  MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioAction("verifyEndpoint", async (state) => {
  try {
    const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
      axios.get(`http://${state.loadBalancerDns}`),
    );
    state.endpointResponse = JSON.stringify(response.data, null, 2);
  } catch (e) {
    state.verifyEndpointError = e;
  }
}),
new ScenarioOutput("verifiedEndpoint", (state) => {
  if (state.verifyEndpointError) {
    console.error(state.verifyEndpointError);
  } else {
    return MESSAGES.verifiedEndpoint.replace(
      "${ENDPOINT_RESPONSE}",
      state.endpointResponse,
    );
  }
}),
];
```

Erstellen Sie Schritte, um die Demo auszuführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
```

```
    DescribeTargetGroupsCommand,  
    DescribeTargetHealthCommand,  
    ElasticLoadBalancingV2Client,  
} from "@aws-sdk/client-elastic-load-balancing-v2";  
import {  
    DescribeInstanceInformationCommand,  
    PutParameterCommand,  
    SSMClient,  
    SendCommandCommand,  
} from "@aws-sdk/client-ssm";  
import {  
    IAMClient,  
    CreatePolicyCommand,  
    CreateRoleCommand,  
    AttachRolePolicyCommand,  
    CreateInstanceProfileCommand,  
    AddRoleToInstanceProfileCommand,  
    waitUntilInstanceProfileExists,  
} from "@aws-sdk/client-iam";  
import {  
    AutoScalingClient,  
    DescribeAutoScalingGroupsCommand,  
    TerminateInstanceInAutoScalingGroupCommand,  
} from "@aws-sdk/client-auto-scaling";  
import {  
    DescribeIamInstanceProfileAssociationsCommand,  
    EC2Client,  
    RebootInstancesCommand,  
    ReplaceIamInstanceProfileAssociationCommand,  
} from "@aws-sdk/client-ec2";  
  
import {  
    ScenarioAction,  
    ScenarioInput,  
    ScenarioOutput,  
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";  
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";  
  
import { MESSAGES, NAMES, RESOURCES_PATH } from "./constants.js";  
import { findLoadBalancer } from "./shared.js";  
  
const getRecommendation = new ScenarioAction(  
    "getRecommendation",  
    async (state) => {
```

```
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
if (loadBalancer) {
  state.loadBalancerDnsName = loadBalancer.DNSName;
  try {
    state.recommendation = (
      await axios.get(`http://${state.loadBalancerDnsName}`)
    ).data;
  } catch (e) {
    state.recommendation = e instanceof Error ? e.message : e;
  }
} else {
  throw new Error(MESSAGES.demoFindLoadBalancerError);
}
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetGroups]
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetGroups]

  // snippet-start:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  // snippet-end:[javascript.v3.wkflw.resilient.DescribeTargetHealth]
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
```



```
"getHealthCheckResult",
/**
 * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
 */
(state) => {
  const status = state.targetHealthDescriptions
    .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
    .join("\n");
  return `Health check:\n${status}`;
},
{ preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);
```

```
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
```

```
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioAction(
```

```

    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
            new DescribeAutoScalingGroupsCommand({
                AutoScalingGroupNames: [NAMES.autoScalingGroupName],
            }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        // snippet-start:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
            new DescribeIamInstanceProfileAssociationsCommand({
                Filters: [
                    { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
                ],
            }),
        );
        // snippet-end:
[javascript.v3.wkflw.resilient.DescribeIamInstanceProfileAssociations]
        state.instanceProfileAssociationId =
            IamInstanceProfileAssociations[0].AssociationId;
        // snippet-start:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            ec2Client.send(
                new ReplaceIamInstanceProfileAssociationCommand({
                    AssociationId: state.instanceProfileAssociationId,
                    IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
                }),
            ),
        );
        // snippet-end:
[javascript.v3.wkflw.resilient.ReplaceIamInstanceProfileAssociation]

        await ec2Client.send(
            new RebootInstancesCommand({
                InstanceIds: [state.targetInstance.InstanceId],
            }),
        );
    }
}

```

```
    }),
  );

  const ssmClient = new SSMClient({});
  await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
    const { InstanceInformationList } = await ssmClient.send(
      new DescribeInstanceInformationCommand({}),
    );

    const instance = InstanceInformationList.find(
      (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
      throw new Error("Instance not found.");
    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
   */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },

```

```
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    })
  ),
);
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
   ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
   ssm').InstanceInformation }} state
   */
```

```
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }),
  new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: `fake-table-${Date.now()}`,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  })
}
```

```
    }),
    new ScenarioAction("resetTable", async () => {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: NAMES.tableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }),
    new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
    healthCheckLoop,
    loadBalancerLoop,
  ];

  async function createSsmOnlyInstanceProfile() {
    const iamClient = new IAMClient({});
    const { Policy } = await iamClient.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.ssmOnlyPolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "ssm_only_policy.json"),
        ),
      }),
    );
    await iamClient.send(
      new CreateRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: { Service: "ec2.amazonaws.com" },
              Action: "sts:AssumeRole",
            },
          ],
        }),
      }),
    );
    await iamClient.send(
      new AttachRolePolicyCommand({
```



```
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: Policy.Arn,
    })),
);
await iamClient.send(
    new AttachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    })),
);
// snippet-start:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    })),
);
await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
// snippet-end:[javascript.v3.wkflw.resilient.CreateInstanceProfile]
await iamClient.send(
    new AddRoleToInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
    })),
);

return InstanceProfile;
}
```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
    EC2Client,
    DeleteKeyPairCommand,
    DeleteLaunchTemplateCommand,
```

```
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "./constants.js";
import { findLoadBalancer } from "./shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
```

```
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  })),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    } else {
      return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
    }
  })),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      );
      unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
      state.deleteKeyPairError = e;
    }
  })),
  new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
      console.error(state.deleteKeyPairError);
      return MESSAGES.deleteKeyPairError.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    } else {
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }
  })),
  new ScenarioAction("detachPolicyFromRole", async (state) => {
```

```
try {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.detachPolicyFromRoleError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`
    );
  } else {
    await client.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: policy.Arn,
      })
    );
  }
} catch (e) {
  state.detachPolicyFromRoleError = e;
}
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
  if (state.detachPolicyFromRoleError) {
    console.error(state.detachPolicyFromRoleError);
    return MESSAGES.detachPolicyFromRoleError
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const policy = await findPolicy(NAMES.instancePolicyName);

  if (!policy) {
    state.deletePolicyError = new Error(
      `Policy ${NAMES.instancePolicyName} not found.`
    );
  } else {
    return client.send(
      new DeletePolicyCommand({
        PolicyArn: policy.Arn,
```

```
    }),
  );
}
}),
new ScenarioOutput("deletePolicyResult", (state) => {
  if (state.deletePolicyError) {
    console.error(state.deletePolicyError);
    return MESSAGES.deletePolicyError.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  } else {
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  } else {
    return MESSAGES.removedRoleFromInstanceProfile
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
```

```
    try {
      const client = new IAMClient({});
      await client.send(
        new DeleteRoleCommand({
          RoleName: NAMES.instanceRoleName,
        }),
      );
    } catch (e) {
      state.deleteInstanceRoleError = e;
    }
  )),
  new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
      console.error(state.deleteInstanceRoleError);
      return MESSAGES.deleteInstanceRoleError.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    } else {
      return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
      );
    }
  )),
  new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
      // snippet-start:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
      const client = new IAMClient({});
      await client.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.instanceProfileName,
        }),
      );
      // snippet-end:[javascript.v3.wkflw.resilient.DeleteInstanceProfile]
    } catch (e) {
      state.deleteInstanceProfileError = e;
    }
  )),
  new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
      console.error(state.deleteInstanceProfileError);
      return MESSAGES.deleteInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
```

```
        NAMES.instanceProfileName,
    );
} else {
    return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
    );
}
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
        await terminateGroupInstances(NAMES.autoScalingGroupName);
        await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
            await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
        });
    } catch (e) {
        state.deleteAutoScalingGroupError = e;
    }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
    if (state.deleteAutoScalingGroupError) {
        console.error(state.deleteAutoScalingGroupError);
        return MESSAGES.deleteAutoScalingGroupError.replace(
            "${AUTO_SCALING_GROUP_NAME}",
            NAMES.autoScalingGroupName,
        );
    } else {
        return MESSAGES.deletedAutoScalingGroup.replace(
            "${AUTO_SCALING_GROUP_NAME}",
            NAMES.autoScalingGroupName,
        );
    }
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
    const client = new EC2Client({});
    try {
        // snippet-start:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
        await client.send(
            new DeleteLaunchTemplateCommand({
                LaunchTemplateName: NAMES.launchTemplateName,
            }),
        );
        // snippet-end:[javascript.v3.wkflw.resilient.DeleteLaunchTemplate]
    } catch (e) {
```

```
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  } else {
    return MESSAGES.deletedLaunchTemplate.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    // snippet-start:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
    // snippet-end:[javascript.v3.wkflw.resilient.DeleteLoadBalancer]
  } catch (e) {
    state.deleteLoadBalancerError = e;
  }
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
});
```



```
    );
  } else {
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  // snippet-start:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      client.send(
        new DeleteTargetGroupCommand({
          TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
      ),
    );
  } catch (e) {
    state.deleteLoadBalancerTargetGroupError = e;
  }
  // snippet-end:[javascript.v3.wkflw.resilient.DeleteTargetGroup]
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
  if (state.deleteLoadBalancerTargetGroupError) {
    console.error(state.deleteLoadBalancerTargetGroupError);
    return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  } else {
    return MESSAGES.deletedLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    );
  }
}),
```

```
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  } else {
    return MESSAGES.detachedSsmOnlyRoleFromProfile
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)

```

```
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    } else {
        return MESSAGES.detachedSsmOnlyCustomRolePolicy
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
        const iamClient = new IAMClient({});
        await iamClient.send(
            new DetachRolePolicyCommand({
                RoleName: NAMES.ssmOnlyRoleName,
                PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
            }),
        );
    } catch (e) {
        state.detachSsmOnlyAWSRolePolicyError = e;
    }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
        console.error(state.detachSsmOnlyAWSRolePolicyError);
        return MESSAGES.detachSsmOnlyAWSRolePolicyError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    } else {
        return MESSAGES.detachedSsmOnlyAWSRolePolicy
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
        const iamClient = new IAMClient({});
        await iamClient.send(
            new DeleteInstanceProfileCommand({
                InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
            }),
        );
    } catch (e) {
        state.deleteSsmOnlyInstanceProfileError = e;
    }
}),
```

```
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
    console.error(state.deleteSsmOnlyInstanceProfileError);
    return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }
}),
new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DeletePolicyCommand({
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyPolicyError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
  if (state.deleteSsmOnlyPolicyError) {
    console.error(state.deleteSsmOnlyPolicyError);
    return MESSAGES.deleteSsmOnlyPolicyError.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  }
}),
new ScenarioAction("deleteSsmOnlyRole", async (state) => {
  try {
    const iamClient = new IAMClient({});
```

```
    await iamClient.send(
      new DeleteRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyRoleError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
  if (state.deleteSsmOnlyRoleError) {
    console.error(state.deleteSsmOnlyRoleError);
    return MESSAGES.deleteSsmOnlyRoleError.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  } else {
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }
}),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
```

```
try {
  await client.send(
    new DeleteAutoScalingGroupCommand({
      AutoScalingGroupName: groupName,
    }),
  );
} catch (err) {
  if (!(err instanceof Error)) {
    throw err;
  } else {
    console.log(err.name);
    throw err;
  }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
```


```
const group = page.AutoScalingGroups.find(
  (g) => g.AutoScalingGroupName === groupName,
);
if (group) {
  return group;
}
}
throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)

- [ReplacelamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Erstellen Sie einen Benutzer und nehmen Sie eine Rolle an


Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer erstellen und eine Rolle annehmen lassen.

 Warning

Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie [AWS IAM Identity Center](#).

- Erstellen Sie einen Benutzer ohne Berechtigungen.
- Erstellen einer Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets für das Konto erteilt.
- Hinzufügen einer Richtlinie, damit der Benutzer die Rolle übernehmen kann.
- Übernehmen Sie die Rolle und listen Sie S3-Buckets mit temporären Anmeldeinformationen auf, und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie einen IAM-Benutzer und eine Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets erteilt. Der Benutzer hat nur Rechte, um die Rolle anzunehmen. Nachdem Sie die Rolle übernommen haben, verwenden Sie temporäre Anmeldeinformationen, um Buckets für das Konto aufzulisten.


```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "test_name";
const policyName = "test_policy";
const roleName = "test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
  }
}
```

```
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    } else {
      throw caught;
    }
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
  // STS).
  // It's not best practice to use access keys. For more information, see https://aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }
}
```

```
}

const {
  AccessKey: { AccessKeyId, SecretAccessKey },
} = createAccessKeyResponse;

let s3Client = new S3Client({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
// thrown while the user and access keys are still stabilizing.
await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
  try {
    return await listBuckets(s3Client);
  } catch (err) {
    if (err instanceof Error && err.name === "InvalidAccessKeyId") {
      throw err;
    }
  }
});

// Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
// is thrown while the user and access keys are still stabilizing.
const { Role } = await retry(
  {
    intervalInMs: 2000,
    maxRetries: 60,
  },
  () =>
    iamClient.send(
      new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
            },
          ],
        }),
      })
    )
  )
);
```

```
        },
        Action: "sts:AssumeRole",
    },
    ],
    }),
    RoleName: roleName,
    )),
    ),
);

if (!Role) {
    throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
    new CreatePolicyCommand({
        PolicyDocument: JSON.stringify({
            Version: "2012-10-17",
            Statement: [
                {
                    Effect: "Allow",
                    Action: ["s3:ListAllMyBuckets"],
                    Resource: "*",
                },
            ],
        }),
        PolicyName: policyName,
    }),
);

if (!listBucketPolicy) {
    throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
    new AttachRolePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
        RoleName: Role.RoleName,
    }),
);

// Assume the role.
```

```
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
        DurationSeconds: 900,
      }),
    ),
);

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 60 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
```

```
        PolicyArn: listBucketPolicy.Arn,
        RoleName: Role.RoleName,
    })),
);

await iamClient.send(
    new DeletePolicyCommand({
        PolicyArn: listBucketPolicy.Arn,
    })),
);

await iamClient.send(
    new DeleteRoleCommand({
        RoleName: Role.RoleName,
    })),
);

await iamClient.send(
    new DeleteAccessKeyCommand({
        UserName: userName,
        AccessKeyId,
    })),
);

await iamClient.send(
    new DeleteUserCommand({
        UserName: userName,
    })),
);
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
    const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

    if (!Buckets) {
        throw new Error("Buckets not listed");
    }

    console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)
 - [DeleteUser](#)
 - [DeleteUserPolicy](#)
 - [DetachRolePolicy](#)
 - [PutUserPolicy](#)

Kinesis-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Kinesis Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Kinesis-Ereignis mit Lambda verwenden. 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;
}
```


Ein Kinesis-Ereignis mit Lambda verwenden. 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;
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von 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;
}
```

Melden von Fehlern Kinesis Kinesis-Batch-Elementen mit Lambda unter Verwendung von 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;
}
```

Lambda-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Lambda verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Lambda

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von Lambda beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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;
};
```

- Einzelheiten zur API finden Sie [ListFunctions](#) in der AWS SDK for JavaScript API-Referenz.

Themen


- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

CreateFunction

Das folgende Codebeispiel zeigt die Verwendung `CreateFunction`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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);
};
```

- Einzelheiten zur API finden Sie [CreateFunction](#) in der AWS SDK for JavaScript API-Referenz.

DeleteFunction

Das folgende Codebeispiel zeigt die Verwendung `DeleteFunction`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteFunction](#) in der AWS SDK for JavaScript API-Referenz.

GetFunction

Das folgende Codebeispiel zeigt die Verwendung `GetFunction`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetFunction](#) in der AWS SDK for JavaScript API-Referenz.

Invoke

Das folgende Codebeispiel zeigt die Verwendung `Invoke`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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 };
};
```

- Weitere API-Informationen finden Sie unter [Invoke](#) in der AWS SDK for JavaScript -API-Referenz.

ListFunctions

Das folgende Codebeispiel zeigt die Verwendung `ListFunctions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
const listFunctions = () => {
```



```
const client = new LambdaClient({});
const command = new ListFunctionsCommand({});

return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ListFunctions](#) in der AWS SDK for JavaScript API-Referenz.

UpdateFunctionCode

Das folgende Codebeispiel zeigt die Verwendung `UpdateFunctionCode`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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);
};
```

- Einzelheiten zur API finden Sie [UpdateFunctionCode](#) in der AWS SDK for JavaScript API-Referenz.

UpdateFunctionConfiguration

Das folgende Codebeispiel zeigt die Verwendung `UpdateFunctionConfiguration`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
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);
};
```

- Einzelheiten zur API finden Sie [UpdateFunctionConfiguration](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erste Schritte mit Funktionen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine IAM-Rolle und eine Lambda-Funktion und laden Sie den Handlercode hoch.
- Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.
- Aktualisieren Sie den Funktionscode und konfigurieren Sie mit einer Umgebungsvariablen.
- Rufen Sie die Funktion mit neuen Parametern auf und erhalten Sie Ergebnisse. Zeigt das zurückgegebene Ausführungsprotokoll an.
- Listen Sie die Funktionen für Ihr Konto auf und bereinigen Sie dann die Ressourcen.

Weitere Informationen zur Verwendung von Lambda finden Sie unter [Erstellen einer Lambda-Funktion mit der Konsole](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle, die Lambda die Berechtigung erteilt, in Protokolle zu schreiben.

```
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);
};
```

Erstellen Sie eine Lambda-Funktion und laden Sie Handlercode hoch.

```
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);
};
```

Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.

```
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 };
};
```

Aktualisieren Sie den Funktionscode und konfigurieren Sie seine Lambda-Umgebung mit einer Umgebungsvariablen.

```
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);
};
```

Listen Sie die Funktionen für Ihr Konto auf.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Löschen Sie die IAM-Rolle für Ihre Lambda-Funktion.

```
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);
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Aufrufen](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Kinesis-Ereignis mit Lambda verwenden. 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;
}
```

Ein Kinesis-Ereignis mit Lambda verwenden. 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;
}
```

Rufen Sie eine Lambda-Funktion von einem DynamoDB-Trigger aus auf

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzlast ab und protokolliert den Inhalt des Datensatzes.

SDK für (v3) JavaScript

 Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. 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)}`);
};
```

Konsumieren eines DynamoDB-Ereignisses mit Lambda unter Verwendung. 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)}`);
};
```

Rufen Sie eine Lambda-Funktion von einem Amazon DocumentDB-Trigger aus auf

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzlast ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden. 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));
};
```

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden 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));
};
```

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung 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);
    }
  };
};
```

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung 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);
  }
};
```

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

```
}  
}
```


Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. 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;  
  }  
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von 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;
}

```

Melden von Fehlern Kinesis Kinesis-Batch-Elementen mit Lambda unter Verwendung von 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;
}

```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Trigger

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von 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: [] };
};
```

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { DynamoDBBatchItemFailure, DynamoDBStreamEvent } from "aws-lambda";

export const handler = async (event: DynamoDBStreamEvent):
  Promise<DynamoDBBatchItemFailure[]> => {

  const batchItemsFailures: DynamoDBBatchItemFailure[] = []
  let curRecordSequenceNumber
```

```
for(const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber

    if(curRecordSequenceNumber) {
        batchItemsFailures.push({
            itemIdentifier: curRecordSequenceNumber
        })
    }
}

return batchItemsFailures
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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}`);
}
```

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von 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.');
```

Amazon Personalize Personalize-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Personalize Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateBatchInferenceJob

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateBatchInferenceJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});
```

```
// Set the batch inference job's parameters.

export const createBatchInferenceJobParam = {
  jobName: 'JOB_NAME',
  jobInput: { /* required */
    s3DataSource: { /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */
    }
  },
  jobOutput: { /* required */
    s3DataDestination: { /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional integer*/
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateBatchInferenceJobCommand(createBatchInferenceJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [CreateBatchInferenceJob](#) in der AWS SDK for JavaScript API-Referenz.

CreateBatchSegmentJob

Das folgende Codebeispiel zeigt die Verwendung `CreateBatchSegmentJob`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: 'NAME',
  jobInput: { /* required */
    s3DataSource: { /* required */
      path: 'INPUT_PATH', /* required */
      // kmsKeyArn: 'INPUT_KMS_KEY_ARN' /* optional */
    }
  },
  jobOutput: { /* required */
    s3DataDestination: { /* required */
      path: 'OUTPUT_PATH', /* required */
      // kmsKeyArn: 'OUTPUT_KMS_KEY_ARN' /* optional */
    }
  },
  roleArn: 'ROLE_ARN', /* required */
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  numResults: 20 /* optional */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
      CreateBatchSegmentJobCommand(createBatchSegmentJobParam));
    console.log("Success", response);
  }
}
```



```
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateBatchSegmentJob](#) in der AWS SDK for JavaScript API-Referenz.

CreateCampaign

Das folgende Codebeispiel zeigt die Verwendung `CreateCampaign`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: 'SOLUTION_VERSION_ARN', /* required */
  name: 'NAME', /* required */
  minProvisionedTPS: 1 /* optional integer */
}

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(new
CreateCampaignCommand(createCampaignParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateCampaign](#) in der AWS SDK for JavaScript API-Referenz.

CreateDataset

Das folgende Codebeispiel zeigt die Verwendung `CreateDataset`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  datasetType: 'DATASET_TYPE', /* required */
  name: 'NAME', /* required */
  schemaArn: 'SCHEMA_ARN' /* required */
}

export const run = async () => {
```

```
try {
  const response = await personalizeClient.send(new
CreateDatasetCommand(createDatasetParam));
  console.log("Success", response);
  return response; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

- Einzelheiten zur API finden Sie [CreateDataset](#) in der AWS SDK for JavaScript API-Referenz.

CreateDatasetExportJob

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetExportJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetExportJobCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the export job parameters.
export const datasetExportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  jobOutput: {
    s3DataDestination: {
      path: 'S3_DESTINATION_PATH' /* required */
      //kmsKeyArn: 'ARN' /* include if your bucket uses AWS KMS for encryption
    }
  }
}
```

```
    },
    jobName: 'NAME', /* required */
    roleArn: 'ROLE_ARN' /* required */
  }

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetExportJobCommand(datasetExportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateDatasetExportJob](#) in der AWS SDK for JavaScript API-Referenz.

CreateDatasetGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});
```

```
// Set the dataset group parameters.
export const createDatasetGroupParam = {
  name: "NAME" /* required */,
};

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(createDatasetGroupParam),
    );
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run(createDatasetGroupParam);
```

Erstellen Sie eine Domain-Datensatzgruppe.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetGroupCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: 'NAME', /* required */
  domain: 'DOMAIN' /* required for a domain dsG, specify ECOMMERCE or
  VIDEO_ON_DEMAND */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateDatasetGroupCommand(domainDatasetGroupParams));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateDatasetGroup](#) unter AWS SDK for JavaScript API-Referenz.

CreateDatasetImportJob

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetImportJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import {CreateDatasetImportJobCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset import job parameters.
export const datasetImportJobParam = {
  datasetArn: 'DATASET_ARN', /* required */
  dataSource: { /* required */
    dataLocation: 'S3_PATH'
  },
  jobName: 'NAME', /* required */
  roleArn: 'ROLE_ARN' /* required */
}

export const run = async () => {
  try {
```

```
    const response = await personalizeClient.send(new
CreateDatasetImportJobCommand(datasetImportJobParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateDatasetImportJob](#) in der AWS SDK for JavaScript API-Referenz.

CreateEventTracker

Das folgende Codebeispiel zeigt die Verwendung `CreateEventTracker`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateEventTrackerCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the event tracker's parameters.
export const createEventTrackerParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
}

export const run = async () => {
```

```
try {
  const response = await personalizeClient.send(new
CreateEventTrackerCommand(createEventTrackerParam));
  console.log("Success", response);
  return response; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

- Einzelheiten zur API finden Sie [CreateEventTracker](#) in der AWS SDK for JavaScript API-Referenz.

CreateFilter

Das folgende Codebeispiel zeigt die Verwendung `CreateFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  name: 'NAME', /* required */
  filterExpression: 'FILTER_EXPRESSION' /*required */
}
```



```
export const run = async () => {
  try {
    const response = await personalizeClient.send(new
CreateFilterCommand(createFilterParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateFilter](#) in der AWS SDK for JavaScript API-Referenz.

CreateRecommender

Das folgende Codebeispiel zeigt die Verwendung `CreateRecommender`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateRecommenderCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the recommender's parameters.
export const createRecommenderParam = {
  name: 'NAME', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  datasetGroupArn: 'DATASET_GROUP_ARN' /* required */
}
```

```
export const run = async () => {
  try {
    const response = await personalizeClient.send(new
CreateRecommenderCommand(createRecommenderParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateRecommender](#) in der AWS SDK for JavaScript API-Referenz.

CreateSchema

Das folgende Codebeispiel zeigt die Verwendung `CreateSchema`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";
```

```
try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema /* required */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
CreateSchemaCommand(createSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Erstellen Sie ein Schema mit einer Domain.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "./libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from 'fs';

let schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = 'TEST' // for unit tests.
```

```
}

// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: 'NAME', /* required */
  schema: mySchema, /* required */
  domain: 'DOMAIN' /* required for a domain dataset group, specify ECOMMERCE or
VIDEO_ON_DEMAND */
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
CreateSchemaCommand(createDomainSchemaParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSchema](#) unter AWS SDK for JavaScript API-Referenz.

CreateSolution

Das folgende Codebeispiel zeigt die Verwendung `CreateSolution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from
"@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: 'DATASET_GROUP_ARN', /* required */
  recipeArn: 'RECIPE_ARN', /* required */
  name: 'NAME' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateSolutionCommand(createSolutionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSolution](#) in der AWS SDK for JavaScript API-Referenz.

CreateSolutionVersion

Das folgende Codebeispiel zeigt die Verwendung `CreateSolutionVersion`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from
  "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
```

```
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: 'SOLUTION_ARN' /* required */
}

export const run = async () => {
  try {
    const response = await personalizeClient.send(new
    CreateSolutionVersionCommand(solutionVersionParam));
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSolutionVersion](#) in der AWS SDK for JavaScript API-Referenz.

Beispiele für Amazon Personalize Events mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Amazon Personalize Events verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt, wie man es benutztPutEvents.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
var putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
```

```
    new PutEventsCommand(putEventsParam),
  );
  console.log("Success!", response);
  return response; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK for JavaScript API-Referenz.

PutItems

Das folgende Codebeispiel zeigt die Verwendung `PutItems`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
character to escape quotes.
var putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /* required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
        "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
```



```

    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

- Einzelheiten zur API finden Sie [PutItems](#) in der AWS SDK for JavaScript API-Referenz.

PutUsers

Das folgende Codebeispiel zeigt die Verwendung `PutUsers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
character to escape quotes.
var putUsersParam = {
  datasetArn: "DATASET_ARN",

```

```
users: [
  {
    userId: "USER_ID",
    properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
  },
],
];
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [PutUsers](#) in der AWS SDK for JavaScript API-Referenz.

Amazon Personalize Runtime-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Personalize Runtime Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

GetPersonalizedRanking

Das folgende Codebeispiel zeigt, wie man es benutzt `GetPersonalizedRanking`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN", /* required */
  userId: 'USER_ID',          /* required */
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"]
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
    GetPersonalizedRankingCommand(getPersonalizedRankingParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [GetPersonalizedRanking](#) in der AWS SDK for JavaScript API-Referenz.

GetRecommendations

Das folgende Codebeispiel zeigt die Verwendung `GetRecommendations`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "./libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',          /* required */
  numResults: 15             /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
  GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Holen Sie sich eine Empfehlung mit einem Filter (benutzerdefinierte Datensatzgruppe).

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: 'RECOMMENDER_ARN', /* required */
  userId: 'USER_ID', /* required */
  numResults: 15 /* optional */
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
  GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Holen Sie sich gefilterte Empfehlungen von einem Empfehlungsgeber, der in einer Domain-Datensatzgruppe erstellt wurde.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from
  "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});
```

```
// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: 'CAMPAIGN_ARN', /* required */
  userId: 'USER_ID',          /* required */
  numResults: 15,             /* optional */
  filterArn: 'FILTER_ARN',    /* required to filter recommendations */
  filterValues: {
    "PROPERTY": "\"VALUE\"" /* Only required if your filter has a placeholder
parameter */
  }
}

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(new
GetRecommendationsCommand(getRecommendationsParam));
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [GetRecommendations](#) unter AWS SDK for JavaScript API-Referenz.

Amazon Pinpoint Pinpoint-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Pinpoint Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

SendMessage

Das folgende Codebeispiel zeigt, wie man es benutztSendMessage.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

Senden Sie eine E-Mail-Nachricht.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "../libs/pinClient.js";

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
var subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";
```

```
// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>
using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding for the subject line and message body of the email.
var charset = "UTF-8";

const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: fromAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
        },
      },
    },
  },
}
```



```
        TextPart: {
          Charset: charset,
          Data: body_text,
        },
      },
    },
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    } else {
      console.log(recipientResult.MessageId);
    }
  } catch (err) {
    console.log(err.message);
  }
};

run();
```

Senden Sie eine SMS-Nachricht.

```
// Import required AWS SDK clients and commands for Node.js
```

```
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "../libs/pinClient.js";

/* The phone number or short code to send the message from. The phone number
   or short code that you specify has to be associated with your Amazon Pinpoint
   account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number. For best results, you should specify the phone
   number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
   Make sure that the SMS channel is enabled for the project or application
   that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
   time-sensitive content, specify TRANSACTIONAL. If you plan to send
   marketing-related content, specify PROMOTIONAL.*/
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
   // varies by country or region. For more information, see
   https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

var senderId = "MySenderId";

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
```

```

    },
  },
  MessageConfiguration: {
    SMSMessage: {
      Body: message,
      Keyword: registeredKeyword,
      MessageType: messageType,
      OriginationNumber: originationNumber,
      SenderId: senderId,
    },
  },
},
};

const run = async () => {
  try {
    const data = await pinClient.send(new SendMessagesCommand(params));
    console.log(
      "Message sent! " +
      data["MessageResponse"]["Result"][destinationNumber]["StatusMessage"],
    );
  } catch (err) {
    console.log(err);
  }
};
run();

```

- Einzelheiten zur API finden Sie [SendMessages](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie eine E-Mail-Nachricht.

```
"use strict";
```

```
const AWS = require("aws-sdk");

// The AWS Region that you want to use to send the email. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/
const aws_region = "us-west-2";

// The "From" address. This address has to be verified in Amazon Pinpoint
// in the region that you use to send email.
const senderAddress = "sender@example.com";

// The address on the "To" line. If your Amazon Pinpoint account is in
// the sandbox, this address also has to be verified.
var toAddress = "recipient@example.com";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
// that you choose.
const appId = "ce796be37f32f178af652b26eexample";

// The subject line of the email.
var subject = "Amazon Pinpoint (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
var body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----
This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in
Node.js.
For more information, see https://aws.amazon.com/sdk-for-node-js/`;

// The body of the email for recipients whose email clients support HTML content.
var body_html = `
<head></head>
<body>
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
  <p>This email was sent with
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint API</a> using the
    <a href='https://aws.amazon.com/sdk-for-node-js/'>
      AWS SDK for JavaScript in Node.js</a>.</p>
</body>
</html>`;

// The character encoding the you want to use for the subject line and
// message body of the email.
```

```
var charset = "UTF-8";

// Specify that you're using a shared credentials file.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: appId,
  MessageRequest: {
    Addresses: {
      [toAddress]: {
        ChannelType: "EMAIL",
      },
    },
    MessageConfiguration: {
      EmailMessage: {
        FromAddress: senderAddress,
        SimpleEmail: {
          Subject: {
            Charset: charset,
            Data: subject,
          },
          HtmlPart: {
            Charset: charset,
            Data: body_html,
          },
          TextPart: {
            Charset: charset,
            Data: body_text,
          },
        },
      },
    },
  },
};

//Try to send the email.
```

```
pinpoint.sendMessage(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
  } else {
    console.log(
      "Email sent! Message ID: ",
      data["MessageResponse"]["Result"]["toAddress"]["MessageId"]
    );
  }
});
```

Senden Sie eine SMS-Nachricht.

```
"use strict";

var AWS = require("aws-sdk");

// The AWS Region that you want to use to send the message. For a list of
// AWS Regions where the Amazon Pinpoint API is available, see
// https://docs.aws.amazon.com/pinpoint/latest/apireference/.
var aws_region = "us-east-1";

// The phone number or short code to send the message from. The phone number
// or short code that you specify has to be associated with your Amazon Pinpoint
// account. For best results, specify long codes in E.164 format.
var originationNumber = "+12065550199";

// The recipient's phone number. For best results, you should specify the
// phone number in E.164 format.
var destinationNumber = "+14255550142";

// The content of the SMS message.
var message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

// The Amazon Pinpoint project/application ID to use when you send this message.
// Make sure that the SMS channel is enabled for the project or application
```

```
// that you choose.
var applicationId = "ce796be37f32f178af652b26eexample";

// The type of SMS message that you want to send. If you plan to send
// time-sensitive content, specify TRANSACTIONAL. If you plan to send
// marketing-related content, specify PROMOTIONAL.
var messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
var registeredKeyword = "myKeyword";

// The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
// https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html
var senderId = "MySenderId";

// Specify that you're using a shared credentials file, and optionally specify
// the profile that you want to use.
var credentials = new AWS.SharedIniFileCredentials({ profile: "default" });
AWS.config.credentials = credentials;

// Specify the region.
AWS.config.update({ region: aws_region });

//Create a new Pinpoint object.
var pinpoint = new AWS.Pinpoint();

// Specify the parameters to pass to the API.
var params = {
  ApplicationId: applicationId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
      },
    },
  },
}
```

```
    },
  },
};

//Try to send the message.
pinpoint.sendMessage(params, function (err, data) {
  // If something goes wrong, print an error message.
  if (err) {
    console.log(err.message);
    // Otherwise, show the unique ID for the message.
  } else {
    console.log(
      "Message sent! " +
      data["MessageResponse"]["Result"]["destinationNumber"]["StatusMessage"]
    );
  }
});
```

- Einzelheiten zur API finden Sie [SendMessages](#) in der AWS SDK for JavaScript API-Referenz.

Amazon Redshift Redshift-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Redshift Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateCluster

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Erstellen Sie den -Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
  one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
  your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
  cluster subnet group to be associated with this cluster. Defaults to 'default' if
  not specified.
```

```
    DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
    Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
  };

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      "Cluster " + data.Cluster.ClusterIdentifier + " successfully created",
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [CreateCluster](#) in der AWS SDK for JavaScript API-Referenz.

DeleteCluster

Das folgende Codebeispiel zeigt die Verwendung `DeleteCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Erstellen Sie den -Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [DeleteCluster](#) in der AWS SDK for JavaScript API-Referenz.

DescribeClusters

Das folgende Codebeispiel zeigt die Verwendung `DescribeClusters`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
```

```
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Beschreiben Sie Ihre Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [DescribeClusters](#) in der AWS SDK for JavaScript API-Referenz.

ModifyCluster

Das folgende Codebeispiel zeigt die Verwendung `ModifyCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Modifizieren Sie einen Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [ModifyCluster](#) unter AWS SDK for JavaScript API-Referenz.

Amazon S3 S3-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon S3 Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon S3

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon S3.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new S3Client({});

export const helloS3 = async () => {
  const command = new ListBucketsCommand({});

  const { Buckets } = await client.send(command);
  console.log("Buckets: ");
  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
  return Buckets;
};
```

- Einzelheiten zur API finden Sie [ListBuckets](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

CopyObject

Das folgende Codebeispiel zeigt die Verwendung `CopyObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Kopieren Sie das Objekt.

```
import { S3Client, CopyObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CopyObjectCommand({
    CopySource: "SOURCE_BUCKET/SOURCE_OBJECT_KEY",
    Bucket: "DESTINATION_BUCKET",
    Key: "NEW_OBJECT_KEY",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [CopyObject](#) in der AWS SDK for JavaScript API-Referenz.

CreateBucket

Das folgende Codebeispiel zeigt die Verwendung `CreateBucket`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Bucket.

```
import { CreateBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new CreateBucketCommand({
    // The name of the bucket. Bucket names are unique and have several other
    // constraints.
    // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
    bucketnamingrules.html
    Bucket: "bucket-name",
  });

  try {
    const { Location } = await client.send(command);
    console.log(`Bucket created with location ${Location}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateBucket](#) in der AWS SDK for JavaScript API-Referenz.

DeleteBucket

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucket`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie den Bucket.

```
import { DeleteBucketCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Delete a bucket.
export const main = async () => {
  const command = new DeleteBucketCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucket](#) in der AWS SDK for JavaScript API-Referenz.

DeleteBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucketPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie die Bucket-Richtlinie.

```
import { DeleteBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// This will remove the policy from the bucket.
export const main = async () => {
  const command = new DeleteBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucketPolicy](#) in der AWS SDK for JavaScript API-Referenz.

DeleteBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucketWebsite`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie die Website-Konfiguration aus dem Bucket.

```
import { DeleteBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Disable static website hosting on the bucket.
export const main = async () => {
  const command = new DeleteBucketWebsiteCommand({
    Bucket: "test-bucket",
  });


  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucketWebsite](#) in der AWS SDK for JavaScript API-Referenz.

DeleteObject

Das folgende Codebeispiel zeigt die Verwendung `DeleteObject`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie ein Objekt.

```
import { DeleteObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectCommand({
    Bucket: "test-bucket",
    Key: "test-key.txt",
  });


  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteObject](#) in der AWS SDK for JavaScript API-Referenz.

DeleteObjects

Das folgende Codebeispiel zeigt die Verwendung `DeleteObjects`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie mehrere Objekte.

```
import { DeleteObjectsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new DeleteObjectsCommand({
    Bucket: "test-bucket",
    Delete: {
      Objects: [{ Key: "object1.txt" }, { Key: "object2.txt" }],
    },
  });

  try {
    const { Deleted } = await client.send(command);
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted objects:`,
    );
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteObjects](#) in der AWS SDK for JavaScript API-Referenz.

GetBucketAcl

Das folgende Codebeispiel zeigt die Verwendung `GetBucketAcl`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die ACL-Berechtigungen ab.

```
import { GetBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketAclCommand({
    Bucket: "test-bucket",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketAcl](#) in der AWS SDK for JavaScript API-Referenz.

GetBucketCors

Das folgende Codebeispiel zeigt die Verwendung `GetBucketCors`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die CORS-Richtlinie für den Bucket ab.

```
import { GetBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketCorsCommand({
```

```
    Bucket: "test-bucket",
  });

  try {
    const { CORSRules } = await client.send(command);
    CORSRules.forEach((cr, i) => {
      console.log(
        `\\nCORSRule ${i + 1}`,
        `\\n${"-" .repeat(10)}`,
        `\\nAllowedHeaders: ${cr.AllowedHeaders.join(" ")}`,
        `\\nAllowedMethods: ${cr.AllowedMethods.join(" ")}`,
        `\\nAllowedOrigins: ${cr.AllowedOrigins.join(" ")}`,
        `\\nExposeHeaders: ${cr.ExposeHeaders.join(" ")}`,
        `\\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketCors](#) in der AWS SDK for JavaScript API-Referenz.

GetBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetBucketPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die Bucket-Richtlinie ab.

```
import { GetBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";
```

```
const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketPolicyCommand({
    Bucket: "test-bucket",
  });

  try {
    const { Policy } = await client.send(command);
    console.log(JSON.parse(Policy));
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketPolicy](#) in der AWS SDK for JavaScript API-Referenz.

GetBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `GetBucketWebsite`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die Website-Konfiguration ab.

```
import { GetBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetBucketWebsiteCommand({
    Bucket: "test-bucket",
  });
};
```



```
try {
  const { ErrorDocument, IndexDocument } = await client.send(command);
  console.log(
    `Your bucket is set up to host a website. It has an error document:`,
    `${ErrorDocument.Key}, and an index document: ${IndexDocument.Suffix}.`,
  );
} catch (err) {
  console.error(err);
}
};
```

- Einzelheiten zur API finden Sie [GetBucketWebsite](#) in der AWS SDK for JavaScript API-Referenz.

GetObject

Das folgende Codebeispiel zeigt die Verwendung `GetObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Laden Sie das Objekt herunter.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new GetObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
  });

  try {
```

```
const response = await client.send(command);
// The Body object also has 'transformToByteArray' and 'transformToWebStream'
methods.
const str = await response.Body.transformToString();
console.log(str);
} catch (err) {
  console.error(err);
}
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetObject](#) in der AWS SDK for JavaScript API-Referenz.

GetObjectLockConfiguration

Das folgende Codebeispiel zeigt die Verwendung `GetObjectLockConfiguration`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {
  GetObjectLockConfigurationCommand,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new GetObjectLockConfigurationCommand({
    Bucket: bucketName,
```

```

    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
  });

  try {
    const { ObjectLockConfiguration } = await client.send(command);
    console.log(`Object Lock Configuration: ${ObjectLockConfiguration}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}

```

- Einzelheiten zur API finden Sie [GetObjectLockConfiguration](#) in der AWS SDK for JavaScript API-Referenz.

GetObjectRetention

Das folgende Codebeispiel zeigt die Verwendung `GetObjectRetention`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import { GetObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName

```

```
* @param {string} objectKey
*/
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
    const { Retention } = await client.send(command);
    console.log(`Object Retention Settings: ${Retention.Status}`);
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Einzelheiten zur API finden Sie [GetObjectRetention](#) in der AWS SDK for JavaScript API-Referenz.

ListBuckets

Das folgende Codebeispiel zeigt die Verwendung `ListBuckets`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie die Buckets auf.

```
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListBucketsCommand({});

  try {
    const { Owner, Buckets } = await client.send(command);
    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }:`
    );
    console.log(`${Buckets.map((b) => ` • ${b.Name}`).join("\n")}`);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListBuckets](#) in der AWS SDK for JavaScript API-Referenz.

ListObjectsV2

Das folgende Codebeispiel zeigt die Verwendung `ListObjectsV2`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie alle Objekte in Ihrem Bucket auf. Wenn es mehr als ein Objekt gibt, `IsTruncated` `NextContinuationToken` wird es verwendet, um über die gesamte Liste zu iterieren.

```
import {
```

```
S3Client,
// This command supersedes the ListObjectsCommand and is the recommended way to
list objects.
ListObjectsV2Command,
} from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new ListObjectsV2Command({
    Bucket: "my-bucket",
    // The default and maximum number of keys returned is 1000. This limits it to
    // one for demonstration purposes.
    MaxKeys: 1,
  });

  try {
    let isTruncated = true;

    console.log("Your bucket contains the following objects:\n");
    let contents = "";


    while (isTruncated) {
      const { Contents, IsTruncated, NextContinuationToken } =
        await client.send(command);
      const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
      contents += contentsList + "\n";
      isTruncated = IsTruncated;
      command.input.ContinuationToken = NextContinuationToken;
    }
    console.log(contents);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie unter [ListObjectsV2](#) in der AWS SDK for JavaScript API-Referenz.

PutBucketAcl

Das folgende Codebeispiel zeigt die Verwendung `PutBucketAcl`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Legen Sie die Bucket-ACL fest.

```
import { PutBucketAclCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Most Amazon S3 use cases don't require the use of access control lists (ACLs).
// We recommend that you disable ACLs, except in unusual circumstances where
// you need to control access for each object individually.
// Consider a policy instead. For more information see https://docs.aws.amazon.com/
// AmazonS3/latest/userguide/bucket-policies.html.
export const main = async () => {
  // Grant a user READ access to a bucket.
  const command = new PutBucketAclCommand({
    Bucket: "test-bucket",
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
            // AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
            latest/userguide/finding-canonical-user-id.html.
            ID: "canonical-id-1",
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
          API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "FULL_CONTROL",
        },
      ],
      Owner: {
        ID: "canonical-id-2",
```

```
    },  
  },  
});  
  
try {  
  const response = await client.send(command);  
  console.log(response);  
} catch (err) {  
  console.error(err);  
}  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketAcl](#) in der AWS SDK for JavaScript API-Referenz.

PutBucketCors

Das folgende Codebeispiel zeigt die Verwendung `PutBucketCors`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Fügen Sie eine CORS-Regel hinzu.

```
import { PutBucketCorsCommand, S3Client } from "@aws-sdk/client-s3";  
  
const client = new S3Client({});  
  
// By default, Amazon S3 doesn't allow cross-origin requests. Use this command  
// to explicitly allow cross-origin requests.  
export const main = async () => {  
  const command = new PutBucketCorsCommand({  
    Bucket: "test-bucket",  
    CORSConfiguration: {  
      CORSRules: [  

```



```
    {
      // Allow all headers to be sent to this bucket.
      AllowedHeaders: ["*"],
      // Allow only GET and PUT methods to be sent to this bucket.
      AllowedMethods: ["GET", "PUT"],
      // Allow only requests from the specified origin.
      AllowedOrigins: ["https://www.example.com"],
      // Allow the entity tag (ETag) header to be returned in the response. The
      ETag header
      // The entity tag represents a specific version of the object. The ETag
      reflects
      // changes only to the contents of an object, not its metadata.
      ExposeHeaders: ["ETag"],
      // How long the requesting browser should cache the preflight response.
      After
      // this time, the preflight request will have to be made again.
      MaxAgeSeconds: 3600,
    },
  ],
},
});

try {
  const response = await client.send(command);
  console.log(response);
} catch (err) {
  console.error(err);
}
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketCors](#) in der AWS SDK for JavaScript API-Referenz.

PutBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `PutBucketPolicy`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Fügen Sie die Richtlinie hinzu.

```
import { PutBucketPolicyCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutBucketPolicyCommand({
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "AllowGetObject",
          // Allow this particular user to call GetObject on any object in this
bucket.
          Effect: "Allow",
          Principal: {
            AWS: "arn:aws:iam::ACCOUNT-ID:user/USERNAME",
          },
          Action: "s3:GetObject",
          Resource: "arn:aws:s3:::BUCKET-NAME/*",
        },
      ],
    }),
    // Apply the preceding policy to this bucket.
    Bucket: "BUCKET-NAME",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketPolicy](#) in der AWS SDK for JavaScript API-Referenz.

PutBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `PutBucketWebsite`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Legen Sie die Website-Konfiguration fest.

```
import { PutBucketWebsiteCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

// Set up a bucket as a static website.
// The bucket needs to be publicly accessible.
export const main = async () => {
  const command = new PutBucketWebsiteCommand({
    Bucket: "test-bucket",
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request that is for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
```

```
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketWebsite](#) in der AWS SDK for JavaScript API-Referenz.

PutObject

Das folgende Codebeispiel zeigt die Verwendung `PutObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Laden Sie das Objekt hoch.

```
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({});

export const main = async () => {
  const command = new PutObjectCommand({
    Bucket: "test-bucket",
    Key: "hello-s3.txt",
    Body: "Hello S3!",
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

```
}  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutObject](#) in der AWS SDK for JavaScript API-Referenz.

PutObjectLegalHold

Das folgende Codebeispiel zeigt die Verwendung `PutObjectLegalHold`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0  
import { fileURLToPath } from "url";  
import { PutObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";  
  
/**  
 * @param {S3Client} client  
 * @param {string} bucketName  
 * @param {string} objectKey  
 */  
export const main = async (client, bucketName, objectKey) => {  
  const command = new PutObjectLegalHoldCommand({  
    Bucket: bucketName,  
    Key: objectKey,  
    LegalHold: {  
      // Set the status to 'ON' to place a legal hold on the object.  
      // Set the status to 'OFF' to remove the legal hold.  
      Status: "ON",  
    },  
    // Optionally, you can provide additional parameters  
    // ChecksumAlgorithm: "ALGORITHM",  
    // ContentMD5: "MD5_HASH",  
  });  
};
```

```
// ExpectedBucketOwner: "ACCOUNT_ID",
// RequestPayer: "requester",
// VersionId: "OBJECT_VERSION_ID",
});

try {
  const response = await client.send(command);
  console.log(
    `Object legal hold status: ${response.$metadata.httpStatusCode}`,
  );
} catch (err) {
  console.error(err);
}
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Einzelheiten zur API finden Sie [PutObjectLegalHold](#) in der AWS SDK for JavaScript API-Referenz.

PutObjectLockConfiguration

Das folgende Codebeispiel zeigt die Verwendung `PutObjectLockConfiguration`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Legt die Objektsperrenkonfiguration eines Buckets fest.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {
```

```

    PutObjectLockConfigurationCommand,
    S3Client,
  } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Object Lock Configuration updated: ${response.$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}

```

Legen Sie die Standardaufbewahrungsdauer eines Buckets fest.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import {

```

```
    PutObjectLockConfigurationCommand,
    S3Client,
  } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 */
export const main = async (client, bucketName) => {
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
      Rule: {
        DefaultRetention: {
          Mode: "GOVERNANCE",
          Years: 3,
        },
      },
    },
  },
  // Optionally, you can provide additional parameters
  // ExpectedBucketOwner: "ACCOUNT_ID",
  // RequestPayer: "requester",
  // Token: "OPTIONAL_TOKEN",
  });

  try {
    const response = await client.send(command);
    console.log(
      `Default Object Lock Configuration updated: ${response.
$metadata.httpStatusCode}`,
    );
  } catch (err) {
    console.error(err);
  }
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME");
}
```


- Einzelheiten zur API finden Sie [PutObjectLockConfiguration](#) unter AWS SDK for JavaScript API-Referenz.

PutObjectRetention

Das folgende Codebeispiel zeigt die Verwendung `PutObjectRetention`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { fileURLToPath } from "url";
import { PutObjectRetentionCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: objectKey,
    BypassGovernanceRetention: false,
    // ChecksumAlgorithm: "ALGORITHM",
    // ContentMD5: "MD5_HASH",
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    Retention: {
      Mode: "GOVERNANCE", // or "COMPLIANCE"
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
    // VersionId: "OBJECT_VERSION_ID",
  });

  try {
```

```
const response = await client.send(command);
console.log(
  `Object Retention settings updated: ${response.$metadata.httpStatusCode}`,
);
} catch (err) {
  console.error(err);
}
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "BUCKET_NAME", "OBJECT_KEY");
}
```

- Einzelheiten zur API finden Sie [PutObjectRetention](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Eine vorsignierte URL erstellen

Das folgende Codebeispiel zeigt, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine vorsignierte URL, um ein Objekt in einen Bucket hochzuladen.

```
import https from "https";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
}
```

```
  } from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

function put(url, data) {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          resolve(responseBody);
        });
      },
    );
    req.on("error", (err) => {
      reject(err);
    });
    req.write(data);
  });
}
```

```
    req.end();
  });
}

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.txt";

  // There are two ways to generate a presigned URL.
  // 1. Use createPresignedUrl without the S3 client.
  // 2. Use getSignedUrl in conjunction with the S3 client and GetObjectCommand.
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    // After you get the presigned URL, you can provide your own file
    // data. Refer to put() above.
    console.log("Calling PUT using presigned URL without client");
    await put(noClientUrl, "Hello World");

    console.log("Calling PUT using presigned URL with client");
    await put(clientUrl, "Hello World");

    console.log("\nDone. Check your S3 console.");
  } catch (err) {
    console.error(err);
  }
};
```

Erstellen Sie eine vorsignierte URL, um ein Objekt aus einem Bucket herunterzuladen.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
```

```
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

export const main = async () => {
  const REGION = "us-east-1";
  const BUCKET = "example_bucket";
  const KEY = "example_file.jpg";

  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });

    const clientUrl = await createPresignedUrlWithClient({
      region: REGION,
      bucket: BUCKET,
      key: KEY,
    });
  }
}
```

```
});

console.log("Presigned URL without client");
console.log(noClientUrl);
console.log("\n");


console.log("Presigned URL with client");
console.log(clientUrl);
} catch (err) {
  console.error(err);
}
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

Eine Webseite erstellen, die Amazon-S3-Objekte auflistet

Im folgenden Codebeispiel wird veranschaulicht, wie Sie Amazon-S3-Objekte auf einer Webseite auflisten.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Der folgende Code ist die relevante React-Komponente, die das AWS SDK aufruft. Eine lauffähige Version der Anwendung, die diese Komponente enthält, finden Sie unter dem vorherigen GitHub Link.

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";
```

```
function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
traffic from
      // this example site. Here's an example configuration that allows all origins.
Don't
      // do this in production.
      // [
      // {
      //   "AllowedHeaders": ["*"],
      //   "AllowedMethods": ["GET"],
      //   "AllowedOrigins": ["*"],
      //   "ExposeHeaders": [],
      // },
      // ]
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      }),
    });
    const command = new ListObjectsCommand({ Bucket: "bucket-name" });
    client.send(command).then(({ Contents }) => setObjects(Contents || []));
  }, []);

  return (
    <div className="App">
      {objects.map((o) => (
        <div key={o.ETag}>{o.Key}</div>
      ))}
    </div>
  );
};
```

```
}  
  
export default App;
```

- Einzelheiten zur API finden Sie unter [ListObjects AWS SDK for JavaScript](#)API-Referenz.

Erste Schritte mit Buckets und Objekten

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Bucket und laden Sie eine Datei in ihn hoch.
- Laden Sie ein Objekt aus einem Bucket herunter.
- Kopieren Sie ein Objekt in einen Unterordner eines Buckets.
- Listen Sie die Objekte in einem Bucket auf.
- Löschen Sie die Bucket-Objekte und den Bucket.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Importieren Sie zunächst alle erforderlichen Module.

```
// Used to check if currently running file is this file.  
import { fileURLToPath } from "url";  
import { readdirSync, readFileSync, writeFileSync } from "fs";  
  
// Local helper utils.  
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";  
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";  
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";  
  
import {  
  S3Client,  
  CreateBucketCommand,
```



```

PutObjectCommand,
ListObjectsCommand,
CopyObjectCommand,
GetObjectCommand,
DeleteObjectsCommand,
DeleteBucketCommand,
} from "@aws-sdk/client-s3";

```

Die vorherigen Importe verweisen auf einige hilfreiche Dienstprogramme. Diese Dienstprogramme befinden sich lokal in dem GitHub Repository, auf das am Anfang dieses Abschnitts verwiesen wird. Zu Ihrer Information finden Sie im Folgenden Implementierungen dieser Dienstprogramme.

```

export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[] }} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {string} prompt
   */
  checkContinue = async (prompt = "") => {
    const prefix = prompt && prompt + " ";
    let ok = await this.confirm({
      message: `${prefix}Continue?`,
    });
    if (!ok) throw new Error("Exiting...");
  };
}

```

```

/**
 * @param {{ message: string }} options
 */
confirm(options) {
  return confirm(options);
}

/**
 * @param {{ message: string, choices: { name: string, value: string }[] }} options
 */
checkbox(options) {
  return checkbox(options);
}
}

export const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n  ${text}\n${rule}\n`;
};

```

Objekte in S3 werden in „Buckets“ gespeichert. Definieren wir eine Funktion zum Erstellen eines neuen Buckets.

```

export const createBucket = async () => {
  const bucketName = await prompter.input({
    message: "Enter a bucket name. Bucket names must be globally unique:",
  });
  const command = new CreateBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log("Bucket created successfully.\n");
  return bucketName;
};

```

Buckets enthalten „Objekte“. Diese Funktion lädt den Inhalt eines Verzeichnisses als Objekte in Ihren Bucket hoch.

```

export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {
  console.log(`Uploading files from ${folderPath}\n`);
  const keys = readdirSync(folderPath);
  const files = keys.map((key) => {

```

```
const filePath = `${folderPath}/${key}`;
const fileContent = readFileSync(filePath);
return {
  Key: key,
  Body: fileContent,
};
});

for (let file of files) {
  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Body: file.Body,
      Key: file.Key,
    }),
  );
  console.log(`${file.Key} uploaded successfully.`);
}
};
```

Überprüfen Sie nach dem Hochladen von Objekten, ob sie korrekt hochgeladen wurden. Sie können `ListObjects` dafür verwenden. Sie werden die Eigenschaft „Key“ verwenden, die Antwort enthält jedoch noch weitere nützliche Eigenschaften.

```
export const listFilesInBucket = async ({ bucketName }) => {
  const command = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(command);
  const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
  console.log("\nHere's a list of files in the bucket:");
  console.log(contentsList + "\n");
};
```

Gelegentlich möchten Sie vielleicht ein Objekt von einem Bucket in einen anderen kopieren. Verwenden Sie dafür den `CopyObject` Befehl.

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });
};
```

```
if (!proceed) {
  return;
} else {
  const copy = async () => {
    try {
      const sourceBucket = await prompter.input({
        message: "Enter source bucket name:",
      });
      const sourceKey = await prompter.input({
        message: "Enter source key:",
      });
      const destinationKey = await prompter.input({
        message: "Enter destination key:",
      });

      const command = new CopyObjectCommand({
        Bucket: destinationBucket,
        CopySource: `${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });
      await s3Client.send(command);
      await copyFileFromBucket({ destinationBucket });
    } catch (err) {
      console.error(`Copy error.`);
      console.error(err);
      const retryAnswer = await prompter.confirm({ message: "Try again?" });
      if (retryAnswer) {
        await copy();
      }
    }
  };
  await copy();
}
};
```

Es gibt keine SDK-Methode zum Abrufen mehrerer Objekte aus einem Bucket. Stattdessen erstellen Sie eine Liste von Objekten, die Sie herunterladen und durchlaufen können.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {
  const { Contents } = await s3Client.send(
    new ListObjectsCommand({ Bucket: bucketName }),
  );
```

```
const path = await prompter.input({
  message: "Enter destination path for files:",
});

for (let content of Contents) {
  const obj = await s3Client.send(
    new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),
  );
  writeFileSync(
    `${path}/${content.Key}`,
    await obj.Body.transformToByteArray(),
  );
}
console.log("Files downloaded successfully.\n");
};
```

Es ist Zeit, Ihre Ressourcen zu bereinigen. Ein Bucket muss leer sein, bevor er gelöscht werden kann. Mit diesen beiden Funktionen leeren und löschen Sie den Bucket.

```
export const emptyBucket = async ({ bucketName }) => {
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });
  const { Contents } = await s3Client.send(listObjectsCommand);
  const keys = Contents.map((c) => c.Key);

  const deleteObjectsCommand = new DeleteObjectsCommand({
    Bucket: bucketName,
    Delete: { Objects: keys.map((key) => ({ Key: key })) },
  });
  await s3Client.send(deleteObjectsCommand);
  console.log(`${bucketName} emptied successfully.\n`);
};

export const deleteBucket = async ({ bucketName }) => {
  const command = new DeleteBucketCommand({ Bucket: bucketName });
  await s3Client.send(command);
  console.log(`${bucketName} deleted successfully.\n`);
};
```

Die Funktion „main“ fasst alles zusammen. Wenn Sie diese Datei direkt ausführen, wird die Funktion „main“ aufgerufen.

```
const main = async () => {
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../resources/sample_files/.sample_media`;

  try {
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));
    console.log("Let's create a bucket.");
    const bucketName = await createBucket();
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("File upload."));
    console.log(
      "I have some default files ready to go. You can edit the source code to
provide your own.",
    );
    await uploadFilesToBucket({
      bucketName,
      folderPath: OBJECT_DIRECTORY,
    });

    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Copy files."));
    await copyFileFromBucket({ destinationBucket: bucketName });
    await listFilesInBucket({ bucketName });
    await prompter.confirm({ message: continueMessage });

    console.log(wrapText("Download files."));
    await downloadFilesFromBucket({ bucketName });

    console.log(wrapText("Clean up."));
    await emptyBucket({ bucketName });
    await deleteBucket({ bucketName });
  } catch (err) {
    console.error(err);
  }
};
```


- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CopyObject](#)

- [CreateBucket](#)
- [DeleteBucket](#)
- [DeleteObjects](#)
- [GetObject](#)
- [ListObjectsV2](#)
- [PutObject](#)

Ruft die Legal-Hold-Konfiguration eines Objekts ab

Das folgende Codebeispiel zeigt, wie die Legal-Hold-Konfiguration eines S3-Buckets abgerufen wird.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import { fileURLToPath } from "url";
import { GetObjectLegalHoldCommand, S3Client } from "@aws-sdk/client-s3";

/**
 * @param {S3Client} client
 * @param {string} bucketName
 * @param {string} objectKey
 */
export const main = async (client, bucketName, objectKey) => {
  const command = new GetObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    // Optionally, you can provide additional parameters
    // ExpectedBucketOwner: "ACCOUNT_ID",
    // RequestPayer: "requester",
    // VersionId: "OBJECT_VERSION_ID",
  });
```

```
try {
  const response = await client.send(command);
  console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
} catch (err) {
  console.error(err);
}
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main(new S3Client(), "DOC-EXAMPLE-BUCKET", "OBJECT_KEY");
}
```

- Einzelheiten zur API finden Sie [GetObjectLegalHold](#) in der AWS SDK for JavaScript API-Referenz.

Amazon S3 S3-Objekte sperren

Das folgende Codebeispiel zeigt, wie Sie mit den Funktionen zum Sperren von Objekten in S3 arbeiten.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

index.js- Einstiegspunkt für den Workflow. Dadurch werden alle Schritte orchestriert. Die Implementierungsdetails für Szenario, ScenarioInput ScenarioOutput, und ScenarioAction finden GitHub Sie unter.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
```



```
    saveState,
  } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "./welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "./setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Object Locking - Deploy",
      [
        welcome(scenarios),
        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
      ]
    )
  }
}
```

```
        createBucketsAction(scenarios, client),
        updateRetention(scenarios),
        confirmUpdateRetention(scenarios),
        exitOnFalse(scenarios, "confirmUpdateRetention"),
        updateRetentionAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        updateLockPolicy(scenarios),
        confirmUpdateLockPolicy(scenarios),
        exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
        updateLockPolicyAction(scenarios, client),
        confirmSetLegalHoldFileEnabled(scenarios),
        setLegalHoldFileEnabledAction(scenarios, client),
        confirmSetRetentionPeriodFileEnabled(scenarios),
        setRetentionPeriodFileEnabledAction(scenarios, client),
        confirmSetLegalHoldFileRetention(scenarios),
        setLegalHoldFileRetentionAction(scenarios, client),
        confirmSetRetentionPeriodFileRetention(scenarios),
        setRetentionPeriodFileRetentionAction(scenarios, client),
        saveState,
    ],
    initialState,
),
demo: new scenarios.Scenario(
    "S3 Object Locking - Demo",
    [loadState, replAction(scenarios, client)],
    initialState,
),
clean: new scenarios.Scenario(
    "S3 Object Locking - Destroy",
    [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
    ],
    initialState,
),
};
};

// Call function if run directly
```

```
import { fileURLToPath } from "url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const objectLockingScenarios = getWorkflowStages(Scenarios);
  Scenarios.parseScenarioArgs(objectLockingScenarios);
}
```

welcome.steps.js- Gibt Willkommensnachrichten an die Konsole aus.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    `Welcome to the Amazon Simple Storage Service (S3) Object Locking Workflow
    Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
    several S3 buckets and files to demonstrate working with S3 locking features.` ,
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };
```

setup.steps.js- Stellen Sie Buckets, Objekte und Dateieinstellungen bereit.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const bucketPrefix = "js-object-locking";

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    `The following buckets will be created:
      ${bucketPrefix}-no-lock with object lock False.
      ${bucketPrefix}-lock-enabled with object lock True.
      ${bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
```

```
*/
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const noLockBucketName = `${bucketPrefix}-no-lock`;
    const lockEnabledBucketName = `${bucketPrefix}-lock-enabled`;
    const retentionBucketName = `${bucketPrefix}-retention-after-creation`;

    await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
    await client.send(
      new CreateBucketCommand({
        Bucket: lockEnabledBucketName,
        ObjectLockEnabledForBucket: true,
      }),
    );
    await client.send(new CreateBucketCommand({ Bucket: retentionBucketName }));

    state.noLockBucketName = noLockBucketName;
    state.lockEnabledBucketName = lockEnabledBucketName;
    state.retentionBucketName = retentionBucketName;
  });

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",
    `The following test files will be created:
      file0.txt in ${bucketPrefix}-no-lock.
      file1.txt in ${bucketPrefix}-no-lock.
      file0.txt in ${bucketPrefix}-lock-enabled.
      file1.txt in ${bucketPrefix}-lock-enabled.
      file0.txt in ${bucketPrefix}-retention-after-creation.
      file1.txt in ${bucketPrefix}-retention-after-creation.` ,
    { preformatted: true },
  );
```

```
);

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.noLockBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
        Bucket: state.lockEnabledBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      }),
    );
    await client.send(
      new PutObjectCommand({
```

```
        Bucket: state.lockEnabledBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
    })),
);
await client.send(
    new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
    })),
);
await client.send(
    new PutObjectCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
    })),
);
});

/**
 * @param {Scenarios} scenarios
 */
const updateRetention = (scenarios) =>
    new scenarios.ScenarioOutput(
        "updateRetention",
        `A bucket can be configured to use object locking with a default retention
period.
A default retention period will be configured for ${bucketPrefix}-retention-
after-creation.` ,
        { preformatted: true },
    );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateRetention = (scenarios) =>
    new scenarios.ScenarioInput(
        "confirmUpdateRetention",
        "Configure default retention period?",
    );
```

```
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.retentionBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
          Rule: {
            DefaultRetention: {
              Mode: "GOVERNANCE",
              Years: 1,
            },
          },
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    `Object lock policies can also be added to existing buckets.
    An object lock policy will be added to ${bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );
```



```
);

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
      ${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
```

```
* @param {Scenarios} scenarios
* @param {S3Client} client
*/
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
      ${state.lockEnabledBucketName}?
      Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
      to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
```

```

const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        })),
      );
      console.log(
        `Set retention for file1.txt in ${state.lockEnabledBucketName} until
        ${retentionDate.toISOString().split("T")[0]}.`,
      );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
      ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>

```

```
new scenarios.ScenarioAction(
  "setLegalHoldFileRetentionAction",
  async (state) => {
    await client.send(
      new PutObjectLegalHoldCommand({
        Bucket: state.retentionBucketName,
        Key: "file0.txt",
        LegalHold: {
          Status: ObjectLockLegalHoldStatus.ON,
        },
      }),
    );
    console.log(
      `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
    );
  },
  { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
);

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
      ${state.retentionBucketName}?
      Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
      to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
```

```
    retentionDate.setDate(retentionDate.getDate() + 1);
    await client.send(
      new PutObjectRetentionCommand({
        Bucket: state.retentionBucketName,
        Key: "file1.txt",
        Retention: {
          Mode: ObjectLockRetentionMode.GOVERNANCE,
          RetainUntilDate: retentionDate,
        },
        BypassGovernanceRetention: true,
      }),
    );
    console.log(
      `Set retention for file1.txt in ${state.retentionBucketName} until
      ${retentionDate.toISOString().split("T")[0]}.`,
    );
  },
  { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
);

export {
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

repl.steps.js- Dateien in den Buckets anzeigen und löschen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    `Explore the S3 locking features by selecting one of the following choices`,
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
      ],
    },
  ),
```

```

    {
      name: "Attempt to delete a file with retention period bypass.",
      value: choices.DELETE_FILE_WITH_RETENTION,
    },
    { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
    {
      name: "View the object and bucket retention settings for a file.",
      value: choices.VIEW_RETENTION_SETTINGS,
    },
    {
      name: "View the legal hold settings for a file.",
      value: choices.VIEW_LEGAL_HOLD_SETTINGS,
    },
    { name: "Finish the workflow.", value: choices.EXIT },
  ],
},
);

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(

```

```
"replAction",
async (state) => {
  const files = await getAllFiles(client, [
    state.noLockBucketName,
    state.lockEnabledBucketName,
    state.retentionBucketName,
  ]);

  const fileInput = new scenarios.ScenarioInput(
    "selectedFile",
    "Select a file:",
    {
      type: "select",
      choices: files.map((file, index) => ({
        name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
          file.version
        })`,
        value: index,
      })),
    },
  );

  const { replChoice } = state;

  switch (replChoice) {
    case choices.LIST_ALL_FILES: {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);
      state.replOutput = files
        .map(
          (file) =>
            `${file.bucket}: ${file.key} (version: ${file.version})`,
        )
        .join("\n");
      break;
    }
    case choices.DELETE_FILE: {
      /** @type {number} */
      const fileToDelete = await fileInput.handle(state);
      const selectedFile = files[fileToDelete];
      try {
```



```
    await client.send(
      new DeleteObjectCommand({
        Bucket: selectedFile.bucket,
        Key: selectedFile.key,
        VersionId: selectedFile.version,
      }),
    );
    state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
  } catch (err) {
    state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
  }
  break;
}
case choices.DELETE_FILE_WITH_RETENTION: {
  /** @type {number} */
  const fileToDelete = await fileInput.handle(state);
  const selectedFile = files[fileToDelete];
  try {
    await client.send(
      new DeleteObjectCommand({
        Bucket: selectedFile.bucket,
        Key: selectedFile.key,
        VersionId: selectedFile.version,
        BypassGovernanceRetention: true,
      }),
    );
    state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
  } catch (err) {
    state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
  }
  break;
}
case choices.OVERWRITE_FILE: {
  /** @type {number} */
  const fileToOverwrite = await fileInput.handle(state);
  const selectedFile = files[fileToOverwrite];
  try {
    await client.send(
      new PutObjectCommand({
        Bucket: selectedFile.bucket,
```

```
        Key: selectedFile.key,
        Body: "New content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
    )),
    );
    state.replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket}`;
    } catch (err) {
        state.replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
    }
    break;
}
case choices.VIEW_RETENTION_SETTINGS: {
    /** @type {number} */
    const fileToView = await fileInput.handle(state);
    const selectedFile = files[fileToView];
    try {
        const retention = await client.send(
            new GetObjectRetentionCommand({
                Bucket: selectedFile.bucket,
                Key: selectedFile.key,
                VersionId: selectedFile.version,
            })),
        );
        const bucketConfig = await client.send(
            new GetObjectLockConfigurationCommand({
                Bucket: selectedFile.bucket,
            })),
        );
        state.replOutput = `Object retention for ${selectedFile.key}
in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
    } catch (err) {
        state.replOutput = `Unable to fetch object lock retention:
'${err.message}'`;
    }
    break;
}
case choices.VIEW_LEGAL_HOLD_SETTINGS: {
```

```

    /** @type {number} */
    const fileToView = await fileInput.handle(state);
    const selectedFile = files[fileToView];
    try {
      const legalHold = await client.send(
        new GetObjectLegalHoldCommand({
          Bucket: selectedFile.bucket,
          Key: selectedFile.key,
          VersionId: selectedFile.version,
        }),
      );
      state.replOutput = `Object legal hold for ${selectedFile.key} in
${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
    } catch (err) {
      state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
    }
    break;
  }
  default:
    throw new Error(`Invalid replChoice: ${replChoice}`);
}
},
{
  whileConfig: {
    whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
    input: replInput(scenarios),
    output: new scenarios.ScenarioOutput(
      "REPL output",
      (state) => state.replOutput,
      { preformatted: true },
    ),
  },
},
);

export { replInput, replAction, choices };

```

`clean.steps.js`- Zerstöre alle erstellten Ressourcen.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {

```

```
DeleteObjectCommand,
DeleteBucketCommand,
ListObjectVersionsCommand,
GetObjectLegalHoldCommand,
GetObjectRetentionCommand,
PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
```

```
objectsResponse = await client.send(
  new ListObjectVersionsCommand({
    Bucket: bucket,
  }),
);
} catch (e) {
  if (e instanceof Error && e.name === "NoSuchBucket") {
    console.log("Object's bucket has already been deleted.");
    continue;
  } else {
    throw e;
  }
}

for (const version of objectsResponse.Versions || []) {
  const { Key, VersionId } = version;

  try {
    const legalHold = await client.send(
      new GetObjectLegalHoldCommand({
        Bucket: bucket,
        Key,
        VersionId,
      }),
    );

    if (legalHold.LegalHold?.Status === "ON") {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: bucket,
          Key,
          VersionId,
          LegalHold: {
            Status: "OFF",
          },
        }),
      );
    }
  } catch (err) {
    console.log(
      `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
    );
  }
}
```

```
    try {
      const retention = await client.send(
        new GetObjectRetentionCommand({
          Bucket: bucket,
          Key,
          VersionId,
        })),
      );

      if (retention.Retention?.Mode === "GOVERNANCE") {
        await client.send(
          new DeleteObjectCommand({
            Bucket: bucket,
            Key,
            VersionId,
            BypassGovernanceRetention: true,
          })),
          );
      }
    } catch (err) {
      console.log(
        `Unable to fetch object lock retention for ${Key} in ${bucket}:
        '${err.message}'`,
      );
    }

    await client.send(
      new DeleteObjectCommand({
        Bucket: bucket,
        Key,
        VersionId,
      })),
      );
  }

  await client.send(new DeleteBucketCommand({ Bucket: bucket }));
  console.log(`Delete for ${bucket} complete.`);
}
});

export { confirmCleanup, cleanupAction };
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.

- [GetObjectLegalHold](#)
- [GetObjectLockConfiguration](#)
- [GetObjectRetention](#)
- [PutObjectLegalHold](#)
- [PutObjectLockConfiguration](#)
- [PutObjectRetention](#)

Hoch- oder Herunterladen großer Dateien

Das folgende Codebeispiel zeigt, wie Sie große Dateien zu und von Amazon S3 hochladen oder herunterladen.

Weitere Informationen finden Sie unter [Hochladen eines Objekts mit Multipart-Upload](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Laden Sie eine große Datei hoch.

```
import {
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
  S3Client,
} from "@aws-sdk/client-s3";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

export const main = async () => {
  const s3Client = new S3Client({});
```

```
const bucketName = "test-bucket";
const key = "multipart.txt";
const str = createString();
const buffer = Buffer.from(str, "utf8");

let uploadId;

try {
  const multipartUpload = await s3Client.send(
    new CreateMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
    }),
  );

  uploadId = multipartUpload.UploadId;

  const uploadPromises = [];
  // Multipart uploads require a minimum size of 5 MB per part.
  const partSize = Math.ceil(buffer.length / 5);

  // Upload each part.
  for (let i = 0; i < 5; i++) {
    const start = i * partSize;
    const end = start + partSize;
    uploadPromises.push(
      s3Client
        .send(
          new UploadPartCommand({
            Bucket: bucketName,
            Key: key,
            UploadId: uploadId,
            Body: buffer.subarray(start, end),
            PartNumber: i + 1,
          }),
        )
        .then((d) => {
          console.log("Part", i + 1, "uploaded");
          return d;
        }),
    );
  }

  const uploadResults = await Promise.all(uploadPromises);
```



```
return await s3Client.send(
  new CompleteMultipartUploadCommand({
    Bucket: bucketName,
    Key: key,
    UploadId: uploadId,
    MultipartUpload: {
      Parts: uploadResults.map(({ ETag }, i) => ({
        ETag,
        PartNumber: i + 1,
      })),
    },
  }),
);

// Verify the output by downloading the file from the Amazon Simple Storage
// Service (Amazon S3) console.
// Because the output is a 25 MB string, text editors might struggle to open the
// file.
} catch (err) {
  console.error(err);

  if (uploadId) {
    const abortCommand = new AbortMultipartUploadCommand({
      Bucket: bucketName,
      Key: key,
      UploadId: uploadId,
    });

    await s3Client.send(abortCommand);
  }
}
};
```

Laden Sie eine große Datei herunter.

```
import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;
```

```
export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: parseInt(start),
    end: parseInt(end),
    length: parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

// When downloading a large file, you might want to break it down into
// smaller pieces. Amazon S3 accepts a Range header to specify the start
// and end of the byte range to be downloaded.
const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };

  while (!isComplete(rangeAndLength)) {
    const { end } = rangeAndLength;
    const nextRange = { start: end + 1, end: end + oneMB };

    console.log(`Downloading bytes ${nextRange.start} to ${nextRange.end}`);

    const { ContentRange, Body } = await getObjectRange({
      bucket,
      key,
```

```
        ...nextRange,
    });

    writeStream.write(await Body.transformToByteArray());
    rangeAndLength = getRangeAndLength(ContentRange);
  }
};

export const main = async () => {
  await downloadInChunks({
    bucket: "my-cool-bucket",
    key: "my-cool-object.txt",
  });
};
```

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung 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);
  }
};
```

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung 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);  
}  
};
```

S3 Glacier-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit S3 Glacier Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen


- [Aktionen](#)

Aktionen

CreateVault

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateVault`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Erstellen des Tresors.

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";


// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
  }
};
run();
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [CreateVault](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });


// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateVault](#) in der AWS SDK for JavaScript API-Referenz.

UploadArchive

Das folgende Codebeispiel zeigt die Verwendung `UploadArchive`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Hochladen des Archivs.

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UploadArchive](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UploadArchive](#) in der AWS SDK for JavaScript API-Referenz.

SageMaker Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden SageMaker.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo SageMaker

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung beginnen SageMaker.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console or with the CreateNotebookInstanceCommand.",
    );
  }
}
```

```
);
} else {
  console.log(
    instances
      .map(
        (i) =>
          `• Instance: ${i.NotebookInstanceName}\n  Arn:${
            i.NotebookInstanceArn
          } \n  Creation Date: ${i.CreationTime.toISOString()}`,
      )
      .join("\n"),
  );
}

return response;
};
```

- Einzelheiten zur API finden Sie [ListNotebookInstances](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreatePipeline

Das folgende Codebeispiel zeigt die Verwendung `CreatePipeline`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Eine Funktion, die mithilfe einer lokal bereitgestellten JSON-Definition eine SageMaker Pipeline erstellt.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 * definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
    implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../workflows/sagemaker_pipelines/resources/
    GeoSpatialPipeline.json`,
  )
  .toString()
  .replace(/\*FUNCTION_ARN\*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
  }
}
```

```
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}
```

- Einzelheiten zur API finden Sie [CreatePipeline](#) unter AWS SDK for JavaScript API-Referenz.

DeletePipeline

Das folgende Codebeispiel zeigt die Verwendung `DeletePipeline`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Die Syntax zum Löschen einer SageMaker Pipeline. Dieser Code ist Teil einer größeren Funktion. Weitere Informationen finden Sie unter „Eine Pipeline erstellen“ oder im GitHub Repository.

```
await sagemakerClient.send(  
  new DeletePipelineCommand({ PipelineName: name } ),  
);
```

- Einzelheiten zur API finden Sie [DeletePipeline](#) in AWS SDK for JavaScript der API-Referenz.

DescribePipelineExecution

Das folgende Codebeispiel zeigt die Verwendung `DescribePipelineExecution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Warten Sie, bis eine SageMaker Pipeline-Ausführung erfolgreich ist, fehlschlägt oder beendet wird.

```
/**  
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or  
 * 'FAILED'.  
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-  
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props  
 */  
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {  
  const command = new DescribePipelineExecutionCommand({  
    PipelineExecutionArn: arn,  
  });  
  
  let complete = false;  
  let intervalInSeconds = 15;  
  const COMPLETION_STATUSES = [  
    PipelineExecutionStatus.FAILED,  
    PipelineExecutionStatus.STOPPED,
```

```
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
        again.`
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}
```

- Einzelheiten zur API finden Sie [DescribePipelineExecution](#) unter AWS SDK for JavaScript API-Referenz.

StartPipelineExecution

Das folgende Codebeispiel zeigt die Verwendung `StartPipelineExecution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Starten Sie eine SageMaker Pipeline-Ausführung.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
   * configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };
}
```



```
    },
  };

  /**
   * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
   requires
   * latitude and longitude values.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobConfig}
   */
  const jobConfig = {
    ReverseGeocodingConfig: {
      XAttributeName: "Longitude",
      YAttributeName: "Latitude",
    },
  };

  const { PipelineExecutionArn } = await sagemakerClient.send(
    new StartPipelineExecutionCommand({
      PipelineName: name,
      PipelineExecutionDisplayName: `${name}-example-execution`,
      PipelineParameters: [
        { Name: "parameter_execution_role", Value: roleArn },
        { Name: "parameter_queue_url", Value: queueUrl },
        {
          Name: "parameter_vej_input_config",
          Value: JSON.stringify(inputConfig),
        },
        {
          Name: "parameter_vej_export_config",
          Value: JSON.stringify(outputConfig),
        },
        {
          Name: "parameter_step_1_vej_config",
          Value: JSON.stringify(jobConfig),
        },
      ],
    }),
  );

  return {
    arn: PipelineExecutionArn,
  };
}
```

- Einzelheiten zur API finden Sie [StartPipelineExecution](#) unter AWS SDK for JavaScript API-Referenz.

Szenarien

Beginnen Sie mit Geodatenjobs und Pipelines

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Richten Sie Ressourcen für eine Pipeline ein.
- Richten Sie eine Pipeline ein, die einen Geodatenauftrag ausführt.
- Pipeline-Ausführung starten.
- Überwachen Sie den Status der Ausführung.
- Sehen Sie sich die Ausgabe der Pipeline an.
- Ressourcen bereinigen.

Weitere Informationen finden Sie unter [SageMaker Pipelines mithilfe von AWS SDKs erstellen und ausführen auf](#) [Community.aws](#).

SDK für (v3) JavaScript

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Der folgende Dateiauszug enthält Funktionen, die den SageMaker Client zur Verwaltung einer Pipeline verwenden.

```
import { readFileSync } from "fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
```

```
DeletePolicyCommand,  
AttachRolePolicyCommand,  
DetachRolePolicyCommand,  
GetRoleCommand,  
ListPoliciesCommand,  
} from "@aws-sdk/client-iam";  
  
import {  
  PublishLayerVersionCommand,  
  DeleteLayerVersionCommand,  
  CreateFunctionCommand,  
  Runtime,  
  DeleteFunctionCommand,  
  CreateEventSourceMappingCommand,  
  DeleteEventSourceMappingCommand,  
  GetFunctionCommand,  
} from "@aws-sdk/client-lambda";  
  
import {  
  PutObjectCommand,  
  CreateBucketCommand,  
  DeleteBucketCommand,  
  DeleteObjectCommand,  
  GetObjectCommand,  
  ListObjectsV2Command,  
} from "@aws-sdk/client-s3";  
  
import {  
  CreatePipelineCommand,  
  DeletePipelineCommand,  
  DescribePipelineCommand,  
  DescribePipelineExecutionCommand,  
  PipelineExecutionStatus,  
  StartPipelineExecutionCommand,  
} from "@aws-sdk/client-sagemaker";  
  
import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-geospatial";  
  
import {  
  CreateQueueCommand,  
  DeleteQueueCommand,  
  GetQueueAttributesCommand,  
  GetQueueUrlCommand,
```

```
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 * props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        })),
    ),
  );

  let role = null;

  try {
    const { Role } = await createRole();
    role = Role;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "EntityAlreadyExistsException"
    ) {
      const { Role } = await iamClient.send(
        new GetRoleCommand({ RoleName: name }),
      );
      role = Role;
    } else {
      throw caught;
    }
  }
}
```

```
    }
  }

  return {
    arn: role.Arn,
    cleanUp: async () => {
      await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
  };
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 * AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 * Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 * pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: [
          "sqs:ReceiveMessage",
          "sqs>DeleteMessage",
          "sqs:GetQueueAttributes",
          "logs>CreateLogGroup",
          "logs>CreateLogStream",
          "logs:PutLogEvents",
          "sagemaker-geospatial:StartVectorEnrichmentJob",
          "sagemaker-geospatial:GetVectorEnrichmentJob",
          "sagemaker:SendPipelineExecutionStepFailure",
          "sagemaker:SendPipelineExecutionStepSuccess",
          "sagemaker-geospatial:ExportVectorEnrichmentJob",
        ],
        Resource: "*",
      },
    ],
  },
```

```
    {
      Effect: "Allow",
      // The AWS Lambda function needs permission to pass the pipeline execution
role to
      // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
function
      // from elevating privileges. For more information, see:
      // https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_use_passrole.html
      Action: ["iam:PassRole"],
      Resource: `${pipelineExecutionRoleArn}`,
      Condition: {
        StringEquals: {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com",
            "sagemaker-geospatial.amazonaws.com",
          ],
        },
      },
    },
  ],
};

const createPolicy = () =>
  iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    }
  }
}
```

```
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}

return {
  arn: policy?.Arn,
  policyConfig,
  cleanUp: async () => {
    await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
  },
};
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        })
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon SageMaker Geospatial clients
```

```

* in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
Amazon SageMaker
* Geospatial client wasn't introduced until v3.221.0.
* @param {{ name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient }} props
*/
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    })),
  );
  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,
        })),
    );
  },
};
}

/**
* Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
pipeline
* execution steps.
* @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
lambda').LambdaClient, layerVersionArn: string}} props
*/
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {

```



```
const lambdaPath = `${dirnameFromMetaUrl(
  import.meta.url,
)}lambda/dist/index.mjs.zip`;

// If a function of the same name already exists, return that
// function's ARN instead. By default this is
// "sagemaker-wkflw-lambda-function", so collisions are
// unlikely.
const createFunction = async () => {
  try {
    return await lambdaClient.send(
      new CreateFunctionCommand({
        Code: {
          ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
        },
        Runtime: Runtime.nodejs18x,
        Handler: "index.handler",
        Layers: [layerVersionArn],
        FunctionName: name,
        Role: roleArn,
      }),
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: name }),
      );
      return Configuration;
    } else {
      throw caught;
    }
  }
};

// Function creation fails if the Role is not ready. This retries
// function creation until it succeeds or it times out.
const { FunctionArn } = await retry(
  { intervalInMs: 1000, maxRetries: 60 },
  createFunction,
);
```

```
return {
  arn: FunctionArn,
  cleanUp: async () => {
    await lambdaClient.send(
      new DeleteFunctionCommand({ FunctionName: name }),
    );
  },
};
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
  const s3Path = `${dirnameFromMetaUrl(
    import.meta.url,
  )}../../../../../workflows/sagemaker_pipelines/resources/latlongtest.csv`;

  await s3Client.send(
    new PutObjectCommand({
      Bucket: bucketName,
      Key: "input/sample_data.csv",
      Body: readFileSync(s3Path),
    }),
  );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
(ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
  let role = null;

  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
```

```
AssumeRolePolicyDocument: JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: ["sts:AssumeRole"],
      Principal: {
        Service: [
          "sagemaker.amazonaws.com",
          "sagemaker-geospatial.amazonaws.com",
        ],
      },
    },
  ],
}),
);

try {
  const { Role } = await createRole();
  role = Role;
  // Wait for the role to be ready.
  await wait(10);
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Role } = await iamClient.send(
      new GetRoleCommand({ RoleName: name }),
    );
    role = Role;
  } else {
    throw caught;
  }
}

return {
  arn: role.Arn,
  cleanUp: async () => {
    await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
  },
};
}
```

```
/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 * messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 * import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
  sqsQueueArn,
  lambdaArn,
  iamClient,
  name,
  s3BucketName,
}) {
  const policyConfig = {
    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Action: ["lambda:InvokeFunction"],
        Resource: lambdaArn,
      },
      {
        Effect: "Allow",
        Action: ["s3:*"],
        Resource: [
          `arn:aws:s3:::${s3BucketName}`,
          `arn:aws:s3:::${s3BucketName}/*`,
        ],
      },
      {
        Effect: "Allow",
        Action: ["sqs:SendMessage"],
        Resource: sqsQueueArn,
      },
    ],
  };

  const createPolicy = () =>
    iamClient.send(
      new CreatePolicyCommand({
        PolicyDocument: JSON.stringify(policyConfig),
      })
    );
}
```

```
        PolicyName: name,
      )),
    );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}

return {
  arn: policy?.Arn,
  policyConfig,
  cleanUp: async () => {
    await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
  },
};
}

/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 * definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
```

```
name,
// Assumes an AWS Lambda function has been created for this pipeline.
functionArn,
sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../../workflows/sagemaker_pipelines/resources/
GeoSpatialPipeline.json`,
  )
  .toString()
  .replace(/.*FUNCTION_ARN*/g, functionArn);

  let arn = null;

  const createPipeline = () =>
    sagemakerClient.send(
      new CreatePipelineCommand({
        PipelineName: name,
        PipelineDefinition: pipelineDefinition,
        RoleArn: roleArn,
      }),
    );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
```

```
        throw caught;
    }
}

return {
    arn,
    cleanUp: async () => {
        await sagemakerClient.send(
            new DeletePipelineCommand({ PipelineName: name }),
        );
    },
};
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
    const createSqsQueue = () =>
        sqsClient.send(
            new CreateQueueCommand({
                QueueName: name,
                Attributes: {
                    DelaySeconds: "5",
                    ReceiveMessageWaitTimeSeconds: "5",
                    VisibilityTimeout: "300",
                },
            }),
        );

    let queueUrl = null;
    try {
        const { QueueUrl } = await createSqsQueue();
        queueUrl = QueueUrl;
    } catch (caught) {
        if (caught instanceof Error && caught.name === "QueueNameExists") {
            const { QueueUrl } = await sqsClient.send(
                new GetQueueUrlCommand({ QueueName: name }),
            );
            queueUrl = QueueUrl;
        } else {
            throw caught;
        }
    }
}
```

```

    }
  }

  const { Attributes } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    () =>
      sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: queueUrl,
          AttributeNames: ["QueueArn"],
        }),
      ),
  );

  return {
    queueUrl,
    queueArn: Attributes.QueueArn,
    cleanUp: async () => {
      await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
    },
  };
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS
 * queue.
 * @param {{
 *   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
 *   lambdaName: string,
 *   queueArn: string,
 *   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
 */
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,

```



```
        FunctionName: lambdaName,
      })),
    );

  try {
    const { UUID } = await createEventSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(
        new GetFunctionCommand({ FunctionName: lambdaName }),
      );

      uuid = eventSourceMappings.find(
        (mapping) =>
          mapping.EventSourceArn === queueArn &&
          mapping.FunctionArn === Configuration.FunctionArn,
      ).UUID;
    } else {
      throw caught;
    }
  }

  return {
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteEventSourceMappingCommand({
          UUID: uuid,
        })),
    },
  }
}
```

```

    );
  },
};
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-
s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(
        { client: s3Client },
        { Bucket: name },
      );
      for await (const page of paginator) {
        const objects = page.Contents;
        if (objects) {
          for (const object of objects) {
            await s3Client.send(
              new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
            );
          }
        }
      }
      await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
    },
  };
}

/**

```

```
* Start the execution of the Amazon SageMaker pipeline. Parameters that are
* passed in are used in the AWS Lambda function.
* @param {{
*   name: string,
*   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
*   roleArn: string,
*   queueUrl: string,
*   s3InputBucketName: string,
* }} props
*/
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {
      S3Data: {
        S3Uri: `s3://${bucketName}/input/sample_data.csv`,
      },
    },
    DocumentType: VectorEnrichmentJobDocumentType.CSV,
  };

  /**
   * The Vector Enrichment Job adds additional data to the source CSV. This
   * configuration points
   * to an Amazon S3 prefix where the output will be stored.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").ExportVectorEnrichmentJobOutputConfig}
   */
  const outputConfig = {
    S3Data: {
      S3Uri: `s3://${bucketName}/output/`,
    },
  };
};
```

```
/**
 * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
requires
 * latitude and longitude values.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
 */
const jobConfig = {
  ReverseGeocodingConfig: {
    XAttributeName: "Longitude",
    YAttributeName: "Latitude",
  },
};

const { PipelineExecutionArn } = await sagemakerClient.send(
  new StartPipelineExecutionCommand({
    PipelineName: name,
    PipelineExecutionDisplayName: `${name}-example-execution`,
    PipelineParameters: [
      { Name: "parameter_execution_role", Value: roleArn },
      { Name: "parameter_queue_url", Value: queueUrl },
      {
        Name: "parameter_vej_input_config",
        Value: JSON.stringify(inputConfig),
      },
      {
        Name: "parameter_vej_export_config",
        Value: JSON.stringify(outputConfig),
      },
      {
        Name: "parameter_step_1_vej_config",
        Value: JSON.stringify(jobConfig),
      },
    ],
  })),
);

return {
  arn: PipelineExecutionArn,
};
}

/**
```

```
* Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
'FAILED'.
* @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
*/
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  let intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
again.`,
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error(`Pipeline was forcefully stopped.`);
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-
s3').S3Client}} param0

```

```
*/
export async function getObject({ bucket, s3Client }) {
  const prefix = "output/";
  const { Contents } = await s3Client.send(
    new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
  );

  if (!Contents.length) {
    throw new Error("No objects found in bucket.");
  }

  // Find the CSV file.
  const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

  if (!outputObject) {
    throw new Error(`No CSV file found in bucket with the prefix "${prefix}.`);
  }

  const { Body } = await s3Client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: outputObject.Key,
    }),
  );

  return Body.transformToString();
}
```

Diese Funktion ist ein Auszug aus einer Datei, die die oben genannten Bibliotheksfunktionen verwendet, um eine SageMaker Pipeline einzurichten, auszuführen und alle erstellten Ressourcen zu löschen.

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
```

```
createSagemakerExecutionPolicy,
createSagemakerPipeline,
createSagemakerRole,
getObject,
startPipelineExecution,
uploadCSVDataToS3,
waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };

  cleanUpFunctions = [];

  /**
   * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
   * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
   * @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
   sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
   sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
   import("@aws-sdk/client-sqs").SQSClient }} clients
   */
  constructor(prompter, logger, clients) {
    this.prompter = prompter;
    this.logger = logger;
    this.clients = clients;
  }

  async run() {
    try {
      await this.startWorkflow();
    }
  }
}
```

```
    } catch (err) {
      console.error(err);
      throw err;
    } finally {
      this.logger.logSeparator();
      const doCleanUp = await this.prompter.confirm({
        message: "Clean up resources?",
      });
      if (doCleanUp) {
        await this.cleanUp();
      }
    }
  }

  async cleanUp() {
    // Run all of the clean up functions. If any fail, we log the error and
    continue.
    // This ensures all clean up functions are run.
    for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
      await retry(
        { intervalInMs: 1000, maxRetries: 60, swallowError: true },
        this.cleanUpFunctions[i],
      );
    }
  }

  async startWorkflow() {
    this.logger.logSeparator(MESSAGES.greetingHeader);
    await this.logger.log(MESSAGES.greeting);

    this.logger.logSeparator();
    await this.logger.log(
      MESSAGES.creatingRole.replace(
        "${ROLE_NAME}",
        this.names.LAMBDA_EXECUTION_ROLE,
      ),
    );

    // Create an IAM role that will be assumed by the AWS Lambda function. This
    function
    // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
    GeoSpatial actions.
    const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
      await createLambdaExecutionRole({
```



```
        name: this.names.LAMBDA_EXECUTION_ROLE,
        iamClient: this.clients.IAM,
    });
    // Add a clean up step to a stack for every resource created.
    this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

    await this.logger.log(
        MESSAGES.roleCreated.replace(
            "${ROLE_NAME}",
            this.names.LAMBDA_EXECUTION_ROLE,
        ),
    );

    this.logger.logSeparator();

    await this.logger.log(
        MESSAGES.creatingRole.replace(
            "${ROLE_NAME}",
            this.names.SAGE_MAKER_EXECUTION_ROLE,
        ),
    );

    // Create an IAM role that will be assumed by the SageMaker pipeline. The
    pipeline
    // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
    const {
        arn: pipelineExecutionRoleArn,
        cleanUp: pipelineExecutionRoleCleanUp,
    } = await createSagemakerRole({
        iamClient: this.clients.IAM,
        name: this.names.SAGE_MAKER_EXECUTION_ROLE,
        wait,
    });
    this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

    await this.logger.log(
        MESSAGES.roleCreated.replace(
            "${ROLE_NAME}",
            this.names.SAGE_MAKER_EXECUTION_ROLE,
        ),
    );

    this.logger.logSeparator();
```

```
// Create an IAM policy that allows the AWS Lambda function to invoke SageMaker APIs.
const {
  arn: lambdaExecutionPolicyArn,
  policy: lambdaPolicy,
  cleanUp: lambdaExecutionPolicyCleanUp,
} = await createLambdaExecutionPolicy({
  name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
  iamClient: this.clients.IAM,
  pipelineExecutionRoleArn,
});
this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

await this.logger.log(
  MESSAGES.attachPolicy
    .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
    .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
);

await this.prompter.checkContinue();

// Attach the Lambda execution policy to the execution role.
const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
  roleName: this.names.LAMBDA_EXECUTION_ROLE,
  policyArn: lambdaExecutionPolicyArn,
  iamClient: this.clients.IAM,
});
this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

await this.logger.log(MESSAGES.policyAttached);

this.logger.logSeparator();

// Create Lambda layer for SageMaker packages.
const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
  await createLambdaLayer({
    name: this.names.LAMBDA_LAYER,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaLayerCleanUp);
```

```
await this.logger.log(
  MESSAGES.creatingFunction.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

// Create the Lambda function with the execution role.
const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
  await createLambdaFunction({
    roleArn: lambdaExecutionRoleArn,
    lambdaClient: this.clients.Lambda,
    name: this.names.LAMBDA_FUNCTION,
    layerVersionArn,
  });
this.cleanUpFunctions.push(lambdaCleanUp);

await this.logger.log(
  MESSAGES.functionCreated.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Create an SQS queue for the SageMaker pipeline.
const {
  queueUrl,
  queueArn,
  cleanUp: queueCleanUp,
} = await createSQSQueue({
  name: this.names.SQS_QUEUE,
  sqsClient: this.clients.SQS,
});
this.cleanUpFunctions.push(queueCleanUp);

await this.logger.log(
  MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);
```

```
this.logger.logSeparator();

await this.logger.log(
  MESSAGES.configuringLambdaSQSEventSource
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Configure the SQS queue as an event source for the Lambda.
const { cleanUp: lambdaSQSEventSourceCleanUp } =
  await configureLambdaSQSEventSource({
    lambdaArn,
    lambdaName: this.names.LAMBDA_FUNCTION,
    queueArn,
    sqsClient: this.clients.SQS,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

await this.logger.log(
  MESSAGES.lambdaSQSEventSourceConfigured
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

this.logger.logSeparator();

// Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
// and send messages to the Amazon SQS queue.
const {
  arn: pipelineExecutionPolicyArn,
  policy: sagemakerPolicy,
  cleanUp: pipelineExecutionPolicyCleanUp,
} = await createSagemakerExecutionPolicy({
  sqsQueueArn: queueArn,
  lambdaArn,
  iamClient: this.clients.IAM,
  name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
});
this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

console.log(JSON.stringify(sagemakerPolicy, null, 2));
```

```
await this.logger.log(
  MESSAGES.attachPolicy
    .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
    .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
);

await this.prompter.checkContinue();

// Attach the SageMaker execution policy to the execution role.
const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
  roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
  policyArn: pipelineExecutionPolicyArn,
  iamClient: this.clients.IAM,
});
this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
// Wait for the role to be ready. If the role is used immediately,
// the pipeline will fail.
await wait(5);

await this.logger.log(MESSAGES.policyAttached);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.creatingPipeline.replace(
    "${PIPELINE_NAME}",
    this.names.SAGE_MAKER_PIPELINE,
  ),
);

// Create the SageMaker pipeline.
const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
  roleArn: pipelineExecutionRoleArn,
  functionArn: lambdaArn,
  sagemakerClient: this.clients.SageMaker,
  name: this.names.SAGE_MAKER_PIPELINE,
});
this.cleanUpFunctions.push(pipelineCleanUp);

await this.logger.log(
  MESSAGES.pipelineCreated.replace(
    "${PIPELINE_NAME}",
    this.names.SAGE_MAKER_PIPELINE,
```

```
    ),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  // Create an S3 bucket for storing inputs and outputs.
  const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
    name: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });
  this.cleanUpFunctions.push(s3BucketCleanUp);

  await this.logger.log(
    MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.uploadingInputData.replace(
      "${BUCKET_NAME}",
      this.names.S3_BUCKET,
    ),
  );

  // Upload CSV Lat/Long data to S3.
  await uploadCSVDataToS3({
    bucketName: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
  });

  await this.logger.log(MESSAGES.inputDataUploaded);

  this.logger.logSeparator();

  await this.prompter.checkContinue(MESSAGES.executePipeline);

  // Execute the SageMaker pipeline.
  const { arn: pipelineExecutionArn } = await startPipelineExecution({
    name: this.names.SAGE_MAKER_PIPELINE,
```

```
    sagemakerClient: this.clients.SageMaker,
    roleArn: pipelineExecutionRoleArn,
    bucketName: this.names.S3_BUCKET,
    queueUrl,
  });

  // Wait for the pipeline execution to finish.
  await waitForPipelineComplete({
    arn: pipelineExecutionArn,
    sagemakerClient: this.clients.SageMaker,
    wait,
  });

  this.logger.logSeparator();

  await this.logger.log(MESSAGES.outputDelay);

  // The getOutput function will throw an error if the output is not
  // found. The retry function will retry a failed function call once
  // ever 10 seconds for 2 minutes.
  const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
    getObject({
      bucket: this.names.S3_BUCKET,
      s3Client: this.clients.S3,
    })),
  );

  this.logger.logSeparator();
  await this.logger.log(MESSAGES.outputDataRetrieved);
  console.log(output.split("\n").slice(0, 6).join("\n"));
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Secrets Manager Manager-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit Secrets Manager verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

GetSecretValue

Das folgende Codebeispiel zeigt, wie man es benutzt `GetSecretValue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
```



```

        SecretId: secretName,
    })),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
//   CreatedDate: 2023-08-08T19:29:51.294Z,
//   Name: 'binary-secret-3873048',
//   SecretBinary: Uint8Array(11) [
//     98, 105, 110, 97, 114,
//     121, 32, 100, 97, 116,
//     97
//   ],
//   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
//   VersionStages: [ 'AWSCURRENT' ]
// }

if (response.SecretString) {
    return response.SecretString;
}

if (response.SecretBinary) {
    return response.SecretBinary;
}
};

```

- Einzelheiten zur API finden Sie [GetSecretValue](#) in der AWS SDK for JavaScript API-Referenz.

Amazon SES SES-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon SES Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

CreateReceiptFilter

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateReceiptFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
        address range in CIDR notation (10.0.0.1/24)).
      }
    }
  });
}
```

```

    Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
addressesOptions.
  },
  /*
    The name of the IP address filter. Only ASCII letters, numbers, underscores,
or dashes.
    Must be less than 64 characters and start and end with a letter or number.
  */
  Name: name,
},
});
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });

  try {
    return await sesClient.send(createReceiptFilterCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- Einzelheiten zur API finden Sie [CreateReceiptFilter](#) in der AWS SDK for JavaScript API-Referenz.

CreateReceiptRule

Das folgende Codebeispiel zeigt die Verwendung `CreateReceiptRule`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,
  name,
  ruleSet,
}) => {
  return new CreateReceiptRuleCommand({
    Rule: {
      Actions: [
        {
          S3Action: {
            BucketName: bucketName,
            ObjectKeyPrefix: "email",
          },
        },
      ],
      Recipients: emailAddresses,
      Enabled: true,
      Name: name,
      ScanEnabled: false,
      TlsPolicy: TlsPolicy.Optional,
    },
    RuleSetName: ruleSet, // Required
  });
};
```

```
const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};
```

- Einzelheiten zur API finden Sie [CreateReceiptRule](#) in der AWS SDK for JavaScript API-Referenz.

CreateReceiptRuleSet

Das folgende Codebeispiel zeigt die Verwendung `CreateReceiptRuleSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};
```

```
const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [CreateReceiptRuleSet](#) in der AWS SDK for JavaScript API-Referenz.

CreateTemplate

Das folgende Codebeispiel zeigt die Verwendung `CreateTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
     system.
    */
  });
};
```

```

    */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};

```

- Einzelheiten zur API finden Sie [CreateTemplate](#) in der AWS SDK for JavaScript API-Referenz.

DeleteIdentity

Das folgende Codebeispiel zeigt die Verwendung `DeleteIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";

const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteIdentity](#) in der AWS SDK for JavaScript API-Referenz.

DeleteReceiptFilter

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```



```
const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptFilter](#) in der AWS SDK for JavaScript API-Referenz.

DeleteReceiptRule

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");
```

```
const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptRule](#) in der AWS SDK for JavaScript API-Referenz.

DeleteReceiptRuleSet

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptRuleSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};
```

```
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptRuleSet](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTemplate

Das folgende Codebeispiel zeigt die Verwendung `DeleteTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);
```

```
    try {
      return await sesClient.send(deleteTemplateCommand);
    } catch (err) {
      console.log("Failed to delete template.", err);
      return err;
    }
  };
```

- Einzelheiten zur API finden Sie [DeleteTemplate](#) in der AWS SDK for JavaScript API-Referenz.

GetTemplate

Das folgende Codebeispiel zeigt die Verwendung `GetTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
    }
  }
}
```

```
        return messageRejectedError;
    }
    throw caught;
}
};
```

- Einzelheiten zur API finden Sie [GetTemplate](#) in der AWS SDK for JavaScript API-Referenz.

ListIdentities

Das folgende Codebeispiel zeigt die Verwendung `ListIdentities`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [ListIdentities](#) in der AWS SDK for JavaScript API-Referenz.

ListReceiptFilters

Das folgende Codebeispiel zeigt die Verwendung `ListReceiptFilters`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Einzelheiten zur API finden Sie [ListReceiptFilters](#) in der AWS SDK for JavaScript API-Referenz.

ListTemplates

Das folgende Codebeispiel zeigt die Verwendung `ListTemplates`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
```

```
const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [ListTemplates](#) in der AWS SDK for JavaScript API-Referenz.

SendBulkTemplatedEmail

Das folgende Codebeispiel zeigt die Verwendung `SendBulkTemplatedEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");
```

```
/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
     * each user
     * to a 'Destination' and provide user specific replacement data to create
     * personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
```



```

    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected} */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- Einzelheiten zur API finden Sie [SendBulkTemplatedEmail](#) in der AWS SDK for JavaScript API-Referenz.

SendEmail

Das folgende Codebeispiel zeigt die Verwendung `SendEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```

import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],

```

```
    ToAddresses: [
      toAddress,
      /* more To-email addresses */
    ],
  },
  Message: {
    /* required */
    Body: {
      /* required */
      Html: {
        Charset: "UTF-8",
        Data: "HTML_FORMAT_BODY",
      },
      Text: {
        Charset: "UTF-8",
        Data: "TEXT_FORMAT_BODY",
      },
    },
    Subject: {
      Charset: "UTF-8",
      Data: "EMAIL_SUBJECT",
    },
  },
  Source: fromAddress,
  ReplyToAddresses: [
    /* more items */
  ],
});
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /* @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
  }
}
```

```
    throw caught;
  }
};
```

- Einzelheiten zur API finden Sie [SendEmail](#) in der AWS SDK for JavaScript API-Referenz.

SendRawEmail

Das folgende Codebeispiel zeigt die Verwendung `SendRawEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Verwenden Sie [nodemailer](#), um eine E-Mail mit einem Anhang zu senden.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses/
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
```

```
});

return new Promise((resolve, reject) => {
  transporter.sendMail(
    {
      from,
      to,
      subject: "Hello World",
      text: "Greetings from Amazon SES!",
      attachments: [{ content: "Hello World!", filename: "hello.txt" }],
    },
    (err, info) => {
      if (err) {
        reject(err);
      } else {
        resolve(info);
      }
    },
  );
});
};
```

- Einzelheiten zur API finden Sie [SendRawEmail](#) in der AWS SDK for JavaScript API-Referenz.

SendTemplatedEmail

Das folgende Codebeispiel zeigt die Verwendung `SendTemplatedEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
```

```
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
```

```
    /** @type { import('@aws-sdk/client-ses').MessageRejected} */
    const messageRejectedError = caught;
    return messageRejectedError;
  }
  throw caught;
}
};
```

- Einzelheiten zur API finden Sie [SendTemplatedEmail](#) in der AWS SDK for JavaScript API-Referenz.

UpdateTemplate

Das folgende Codebeispiel zeigt die Verwendung `UpdateTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};
```

```
const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [UpdateTemplate](#) in der AWS SDK for JavaScript API-Referenz.

VerifyDomainIdentity

Das folgende Codebeispiel zeigt die Verwendung `VerifyDomainIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};
```

```
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [VerifyDomainIdentity](#) in der AWS SDK for JavaScript API-Referenz.

VerifyEmailIdentity

Das folgende Codebeispiel zeigt die Verwendung `VerifyEmailIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
```



```
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [VerifyEmailIdentity](#) in der AWS SDK for JavaScript API-Referenz.

Amazon SNS SNS-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon SNS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon SNS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon SNS.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Initialisieren Sie einen SNS-Client und listen Sie Themen in Ihrem Konto auf.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`
  );
  console.log(topics.map((t) => ` * ${t.TopicArn}`).join("\n"));
};
```

- Einzelheiten zur API finden Sie [ListTopics](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

CheckIfPhoneNumberIsOptedOut

Das folgende Codebeispiel zeigt die Verwendung `CheckIfPhoneNumberIsOptedOut`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
//     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   isOptedOut: false
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CheckIfPhoneNumbersOptedOut](#) in der AWS SDK for JavaScript API-Referenz.

ConfirmSubscription

Das folgende Codebeispiel zeigt die Verwendung `ConfirmSubscription`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                        that are not AWS services (HTTP/S, email) need to be
 *                        confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 *                            subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
  xxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ConfirmSubscription](#) in der AWS SDK for JavaScript API-Referenz.

CreateTopic

Das folgende Codebeispiel zeigt die Verwendung `CreateTopic`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//     totalRetryDelay: 0
//   },
//   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME'
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateTopic](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTopic

Das folgende Codebeispiel zeigt die Verwendung `DeleteTopic`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
```

```
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteTopic](#) in der AWS SDK for JavaScript API-Referenz.

GetSMSAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetSMSAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```


Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );


  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Details zu API finden Sie unter [GetSMSAttributes](#) in der AWS SDK for JavaScript -API-Referenz.

GetTopicAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetTopicAttributes`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```

// Attributes: {
//   Policy: '{...}',
//   Owner: 'xxxxxxxxxxxxx',
//   SubscriptionsPending: '1',
//   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
//   TracingConfig: 'PassThrough',
//   EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetries":0,"headerContentType":"text/plain; charset=UTF-8"}}}',
//   SubscriptionsConfirmed: '0',
//   DisplayName: '',
//   SubscriptionsDeleted: '1'
// }
// }
return response;
};

```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetTopicAttributes](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```

// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

```

```
// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetTopicAttributes](#) in der AWS SDK for JavaScript API-Referenz.

ListSubscriptions

Das folgende Codebeispiel zeigt die Verwendung `ListSubscriptions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";
```

```
/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );


  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListSubscriptions](#) in der AWS SDK for JavaScript API-Referenz.

ListTopics

Das folgende Codebeispiel zeigt die Verwendung `ListTopics`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [ListTopics](#) in der AWS SDK for JavaScript API-Referenz.

Publish

Das folgende Codebeispiel zeigt die Verwendung `Publish`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
plain string or an object
 *
 * if you are using the `json`
`MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
```

```
const response = await snsClient.send(
  new PublishCommand({
    Message: message,
    TopicArn: topicArn,
  }),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
// }
return response;
};
```

Veröffentlichen Sie eine Nachricht zu einem Thema mit Gruppen-, Duplizierungs- und Attributoptionen.

```
async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId, deduplicationId, choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });
  }

  if (this.autoDedup === false) {
    await this.logger.log(MESSAGES.deduplicationIdNotice);
    deduplicationId = await this.prompter.input({
      message: MESSAGES.deduplicationIdPrompt,
    });
  }
}
```



```
    }

    choices = await this.prompter.checkbox({
      message: MESSAGES.messageAttributesPrompt,
      choices: toneChoices,
    });
  }

  await this.snsClient.send(
    new PublishCommand({
      TopicArn: this.topicArn,
      Message: message,
      ...(groupId
        ? {
            MessageGroupId: groupId,
          }
        : {}),
      ...(deduplicationId
        ? {
            MessageDeduplicationId: deduplicationId,
          }
        : {}),
      ...(choices
        ? {
            MessageAttributes: {
              tone: {
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    })),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });

  if (publishAnother) {
    await this.publishMessages();
  }
}
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Details zu API finden Sie unter [Veröffentlichen](#) in der AWS SDK for JavaScript -API-Referenz.

SetSMSAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetSMSAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.

```

```
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
    },
 )),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Details zu API finden Sie unter [SetSMSAttributes](#) in der AWS SDK for JavaScript -API-Referenz.

SetTopicAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetTopicAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SetTopicAttributes](#) in der AWS SDK for JavaScript API-Referenz.

Subscribe

Das folgende Codebeispiel zeigt die Verwendung `Subscribe`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
```

```

    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};

```

Abonnieren Sie eine mobile Anwendung für ein Thema.

```

import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 * created
 *
 *                               when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,

```

```
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

Abonnieren Sie eine Lambda-Funktion für ein Thema.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
```

```
// }  
return response;  
};
```

Abonnieren Sie eine SQS-Warteschlange für ein Thema.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";  
  
const client = new SNSClient({});  
  
export const subscribeQueue = async (  
  topicArn = "TOPIC_ARN",  
  queueArn = "QUEUE_ARN",  
) => {  
  const command = new SubscribeCommand({  
    TopicArn: topicArn,  
    Protocol: "sqs",  
    Endpoint: queueArn,  
  });  
  
  const response = await client.send(command);  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,  
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   },  
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-  
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'  
  // }  
  return response;  
};
```

Abonnieren Sie ein Thema mit einem Filter.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";
```



```
const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
      // to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
  // test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Details zu API finden Sie unter [Abonnieren](#) in der AWS SDK for JavaScript -API-Referenz.

Unsubscribe

Das folgende Codebeispiel zeigt die Verwendung `Unsubscribe`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
```

```
//     statusCode: 200,  
//     requestId: '0178259a-9204-507c-b620-78a7570a44c6',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Details zu API finden Sie unter [Abmelden](#) in der AWS SDK for JavaScript -API-Referenz.

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Dies ist der Einstiegspunkt für diesen Workflow.

```
import { SNSClient } from "@aws-sdk/client-sns";  
import { SQSClient } from "@aws-sdk/client-sqs";
```

```
import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Der obige Code stellt die erforderlichen Abhängigkeiten bereit und startet den Workflow. Der nächste Abschnitt enthält den Großteil des Beispiels.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
```

```
* @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
*/
queues = [];
prompter;

/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../..../libs/prompter.js').Prompter} prompter
 * @param {import('../..../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
}
```

```
if (this.isFifo) {
  this.topicName += ".fifo";
  this.logger.logSeparator(MESSAGES.headerFifoNaming);
  await this.logger.log(MESSAGES.appendFifoNotice);
}

const response = await this.snsClient.send(
  new CreateTopicCommand({
    Name: this.topicName,
    Attributes: {
      FifoTopic: this.isFifo ? "true" : "false",
      ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
    },
  }),
);

this.topicArn = response.TopicArn;

await this.logger.log(
  MESSAGES.topicCreatedNotice
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TOPIC_ARN}", this.topicArn),
);
}

async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  let maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });
  });

  if (this.isFifo) {
    queueName += ".fifo";
    await this.logger.log(MESSAGES.appendFifoNotice);
  }
}
```

```
const response = await this.sqsClient.send(
  new CreateQueueCommand({
    QueueName: queueName,
    Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
  }),
);

const { Attributes } = await this.sqsClient.send(
  new GetQueueAttributesCommand({
    QueueUrl: response.QueueUrl,
    AttributeNames: ["QueueArn"],
  }),
);

this.queues.push({
  queueName,
  queueArn: Attributes.QueueArn,
  queueUrl: response.QueueUrl,
});

await this.logger.log(
  MESSAGES.queueCreatedNotice
    .replace("${QUEUE_NAME}", queueName)
    .replace("${QUEUE_URL}", response.QueueUrl)
    .replace("${QUEUE_ARN}", Attributes.QueueArn),
);
}
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              }
            }
          }
        ]
      }
    );
  }
}
```

```
        },
      },
    ],
  },
  null,
  2,
);

if (index !== 0) {
  this.logger.logSeparator();
}

await this.logger.log(MESSAGES.attachPolicyNotice);
console.log(policy);
const addPolicy = await this.prompter.confirm({
  message: MESSAGES.addPolicyConfirmation.replace(
    "${QUEUE_NAME}",
    queue.queueName,
  ),
});

if (addPolicy) {
  await this.sqsClient.send(
    new SetQueueAttributesCommand({
      QueueUrl: queue.queueUrl,
      Attributes: {
        Policy: policy,
      },
    }),
  );
  queue.policy = policy;
} else {
  await this.logger.log(
    MESSAGES.policyNotAttachedNotice.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  );
}
}
}

async subscribeQueuesToTopic() {
```



```
for (const [index, queue] of this.queues.entries()) {
  /**
   * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
   */
  const subscribeParams = {
    TopicArn: this.topicArn,
    Protocol: "sqs",
    Endpoint: queue.queueArn,
  };
  let tones = [];

  if (this.isFifo) {
    if (index === 0) {
      await this.logger.log(MESSAGES.fifoFilterNotice);
    }
    tones = await this.prompter.checkbox({
      message: MESSAGES.fifoFilterSelect.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
      choices: toneChoices,
    });
  }

  if (tones.length) {
    subscribeParams.Attributes = {
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        tone: tones,
      }),
    };
  }
}

const { SubscriptionArn } = await this.snsClient.send(
  new SubscribeCommand(subscribeParams),
);

this.subscriptionArns.push(SubscriptionArn);

await this.logger.log(
  MESSAGES.queueSubscribedNotice
    .replace("${QUEUE_NAME}", queue.queueName)
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
);
```

```
    );  
  }  
}  
  
async publishMessages() {  
  const message = await this.prompter.input({  
    message: MESSAGES.publishMessagePrompt,  
  });  
  
  let groupId, deduplicationId, choices;  
  
  if (this.isFifo) {  
    await this.logger.log(MESSAGES.groupIdNotice);  
    groupId = await this.prompter.input({  
      message: MESSAGES.groupIdPrompt,  
    });  
  
    if (this.autoDedup === false) {  
      await this.logger.log(MESSAGES.deduplicationIdNotice);  
      deduplicationId = await this.prompter.input({  
        message: MESSAGES.deduplicationIdPrompt,  
      });  
    }  
  
    choices = await this.prompter.checkbox({  
      message: MESSAGES.messageAttributesPrompt,  
      choices: toneChoices,  
    });  
  }  
  
  await this.snsClient.send(  
    new PublishCommand({  
      TopicArn: this.topicArn,  
      Message: message,  
      ...(groupId  
        ? {  
          MessageGroupId: groupId,  
        }  
        : {}),  
      ...(deduplicationId  
        ? {  
          MessageDeduplicationId: deduplicationId,  
        }  
        : {}),  
    })  
  );  
}
```

```
    ...(choices
      ? {
        MessageAttributes: {
          tone: {
            DataType: "String.Array",
            StringValue: JSON.stringify(choices),
          },
        },
      }
      : {}),
  )),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});

if (publishAnother) {
  await this.publishMessages();
}

}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      })),
  );

  if (Messages) {
    await this.logger.log(
      MESSAGES.messagesReceivedNotice.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    );
  }

  console.log(Messages);

  await this.sqsClient.send(
    new DeleteMessageBatchCommand({
      QueueUrl: queue.queueUrl,
      Entries: Messages.map((message) => ({
        Id: message.MessageId,
      })),
    })),
  );
}
```

```
        ReceiptHandle: message.ReceiptHandle,
      })),
    })),
  );
} else {
  await this.logger.log(
    MESSAGES.noMessagesReceivedNotice.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  );
}
}

const deleteAndPoll = await this.prompter.confirm({
  message: MESSAGES.deleteAndPollConfirmation,
});

if (deleteAndPoll) {
  await this.receiveAndDeleteMessages();
}
}

async destroyResources() {
  for (const subscriptionArn of this.subscriptionArns) {
    await this.snsClient.send(
      new UnsubscribeCommand({ SubscriptionArn: subscriptionArn })),
    );
  }

  for (const queue of this.queues) {
    await this.sqsClient.send(
      new DeleteQueueCommand({ QueueUrl: queue.queueUrl })),
    );
  }

  if (this.topicArn) {
    await this.snsClient.send(
      new DeleteTopicCommand({ TopicArn: this.topicArn })),
    );
  }
}

async start() {
```

```
console.clear();

try {
  this.logger.logSeparator(MESSAGES.headerWelcome);
  await this.welcome();
  this.logger.logSeparator(MESSAGES.headerFifo);
  await this.confirmFifo();
  this.logger.logSeparator(MESSAGES.headerCreateTopic);
  await this.createTopic();
  this.logger.logSeparator(MESSAGES.headerCreateQueues);
  await this.createQueues();
  this.logger.logSeparator(MESSAGES.headerAttachPolicy);
  await this.attachQueueIamPolicies();
  this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
  await this.subscribeQueuesToTopic();
  this.logger.logSeparator(MESSAGES.headerPublishMessage);
  await this.publishMessages();
  this.logger.logSeparator(MESSAGES.headerReceiveMessages);
  await this.receiveAndDeleteMessages();
} catch (err) {
  console.error(err);
} finally {
  await this.destroyResources();
}
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)

- [Unsubscribe](#)

Serverless-Beispiele

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Amazon SQS SQS-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon SQS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon SQS

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon SQS beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Initialisieren Sie einen Amazon SQS SQS-Client und listen Sie Warteschlangen auf.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = async () => {
  // The configuration object ( `{}` ) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`
  );
}
```



```
);  
console.log(queues.map((t) => ` * ${t}`)).join("\n");  
};
```

- Einzelheiten zur API finden Sie unter [ListQueues](#)API-Referenz.AWS SDK for JavaScript

Themen

- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Aktionen

ChangeMessageVisibility

Das folgende Codebeispiel zeigt die Verwendung `ChangeMessageVisibility`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie eine Amazon SQS SQS-Nachricht und ändern Sie deren Timeout-Sichtbarkeit.

```
import {  
  ReceiveMessageCommand,  
  ChangeMessageVisibilityCommand,  
  SQSClient,  
} from "@aws-sdk/client-sqs";  
  
const client = new SQSClient({});  
const SQS_QUEUE_URL = "queue_url";  
  
const receiveMessage = (queueUrl) =>  
  client.send(  
    new ReceiveMessageCommand({
```

```
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    WaitTimeSeconds: 1,
  })),
);

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie unter [ChangeMessageVisibility AWS SDK for JavaScript API-Referenz](#).

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie eine Amazon SQS SQS-Nachricht und ändern Sie deren Timeout-Sichtbarkeit.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });
```

```
var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};


sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [ChangeMessageVisibility AWS SDK for JavaScriptAPI-Referenz](#).

CreateQueue

Das folgende Codebeispiel zeigt die VerwendungCreateQueue.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Amazon SQS SQS-Standardwarteschlange.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Erstellen Sie eine Amazon SQS SQS-Warteschlange mit langen Abfragen.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
```

```
    // When the wait time for the ReceiveMessage API action is greater than 0,  
    // long polling is in effect. The maximum long polling wait time is 20  
    // seconds. Long polling helps reduce the cost of using Amazon SQS by,  
    // eliminating the number of empty responses and false empty responses.  
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/  
SQSDeveloperGuide/sqs-short-and-long-polling.html  
    ReceiveMessageWaitTimeSeconds: "20",  
  },  
  }  
);  
console.log(response);  
return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [CreateQueue AWS SDK for JavaScript](#)API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie eine Amazon SQS SQS-Standardwarteschlange.

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create an SQS service object  
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });  
  
var params = {  
  QueueName: "SQS_QUEUE_NAME",  
  Attributes: {  
    DelaySeconds: "60",  
    MessageRetentionPeriod: "86400",  
  },  
};
```

```
sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Erstellen Sie eine Amazon SQS SQS-Warteschlange, die auf den Eingang einer Nachricht wartet.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};


sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [CreateQueue AWS SDK for JavaScript API-Referenz](#).

DeleteMessage

Das folgende Codebeispiel zeigt die Verwendung `DeleteMessage`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen und löschen Sie Amazon SQS SQS-Nachrichten.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    })
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
```

```
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
    })),
    );
} else {
    await client.send(
        new DeleteMessageBatchCommand({
            QueueUrl: queueUrl,
            Entries: Messages.map((message) => ({
                Id: message.MessageId,
                ReceiptHandle: message.ReceiptHandle,
            })),
        })),
    );
}
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen und löschen Sie Amazon SQS SQS-Nachrichten.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 10,
```



```
    MessageAttributeNames: ["All"],
    QueueUrl: queueURL,
    VisibilityTimeout: 20,
    WaitTimeSeconds: 0,
  };

  sqs.receiveMessage(params, function (err, data) {
    if (err) {
      console.log("Receive Error", err);
    } else if (data.Messages) {
      var deleteParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
      };
      sqs.deleteMessage(deleteParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Message Deleted", data);
        }
      });
    }
  });
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK for JavaScript API-Referenz.

DeleteMessageBatch

Das folgende Codebeispiel zeigt die Verwendung `DeleteMessageBatch`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import {
```

```
    ReceiveMessageCommand,  
    DeleteMessageCommand,  
    SQSClient,  
    DeleteMessageBatchCommand,  
  } from "@aws-sdk/client-sqs";  
  
const client = new SQSClient({});  
const SQS_QUEUE_URL = "queue_url";  
  
const receiveMessage = (queueUrl) =>  
  client.send(  
    new ReceiveMessageCommand({  
      AttributeNames: ["SentTimestamp"],  
      MaxNumberOfMessages: 10,  
      MessageAttributeNames: ["All"],  
      QueueUrl: queueUrl,  
      WaitTimeSeconds: 20,  
      VisibilityTimeout: 20,  
    }  
  ),  
  );  
  
export const main = async (queueUrl = SQS_QUEUE_URL) => {  
  const { Messages } = await receiveMessage(queueUrl);  
  
  if (!Messages) {  
    return;  
  }  
  
  if (Messages.length === 1) {  
    console.log(Messages[0].Body);  
    await client.send(  
      new DeleteMessageCommand({  
        QueueUrl: queueUrl,  
        ReceiptHandle: Messages[0].ReceiptHandle,  
      }  
    ),  
    );  
  } else {  
    await client.send(  
      new DeleteMessageBatchCommand({  
        QueueUrl: queueUrl,  
        Entries: Messages.map((message) => ({  
          Id: message.MessageId,  
          ReceiptHandle: message.ReceiptHandle,  
        })),  
    ),  
  );  
  }  
}
```

```
    }),  
  );  
}  
};
```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK for JavaScript API-Referenz.

DeleteQueue

Das folgende Codebeispiel zeigt die Verwendung `DeleteQueue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie eine Amazon SQS SQS-Warteschlange.

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";  
  
const client = new SQSClient({});  
const SQS_QUEUE_URL = "test-queue-url";  
  
export const main = async (queueUrl = SQS_QUEUE_URL) => {  
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });  
  
  const response = await client.send(command);  
  console.log(response);  
  return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteQueue](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen Sie eine Amazon SQS SQS-Warteschlange.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};


sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteQueue](#) unter AWS SDK for JavaScript API-Referenz.

GetQueueAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetQueueAttributes`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK for JavaScript API-Referenz.

GetQueueUrl

Das folgende Codebeispiel zeigt die Verwendung `GetQueueUrl`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die URL für eine Amazon SQS SQS-Warteschlange ab.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetQueueUrl](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie die URL für eine Amazon SQS SQS-Warteschlange ab.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetQueueUrl](#)unter AWS SDK for JavaScript API-Referenz.

ListQueues

Das folgende Codebeispiel zeigt die Verwendung `ListQueues`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie Ihre Amazon SQS SQS-Warteschlangen auf.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
```

```
export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    urls.forEach((url) => console.log(url));
  }

  return urls;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Listen Sie Ihre Amazon SQS SQS-Warteschlangen auf.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```



```
}  
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK for JavaScript API-Referenz.

ReceiveMessage

Das folgende Codebeispiel zeigt die Verwendung `ReceiveMessage`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie eine Nachricht aus einer Amazon SQS SQS-Warteschlange.

```
import {  
  ReceiveMessageCommand,  
  DeleteMessageCommand,  
  SQSClient,  
  DeleteMessageBatchCommand,  
} from "@aws-sdk/client-sqs";  
  
const client = new SQSClient({});  
const SQS_QUEUE_URL = "queue_url";  
  
const receiveMessage = (queueUrl) =>  
  client.send(  
    new ReceiveMessageCommand({  
      AttributeNames: ["SentTimestamp"],  
      MaxNumberOfMessages: 10,  
      MessageAttributeNames: ["All"],  
      QueueUrl: queueUrl,  
      WaitTimeSeconds: 20,  
      VisibilityTimeout: 20,  
    })  
  ),
```

```
);

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon SQS SQS-Warteschlange.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
```

```
MessageAttributeNames: ["All"],
QueueUrl: queueUrl,
// The duration (in seconds) for which the call waits for a message
// to arrive in the queue before returning. If a message is available,
// the call returns sooner than WaitTimeSeconds. If no messages are
// available and the wait time expires, the call returns successfully
// with an empty list of messages.
// https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
WaitTimeSeconds: 20,
});

const response = await client.send(command);
console.log(response);
return response;
};
```

- Einzelheiten zur API finden Sie unter [ReceiveMessage AWS SDK for JavaScript API-Referenz](#).

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon SQS SQS-Warteschlange.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
```

```
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueURL,
    WaitTimeSeconds: 20,
  };

  sqs.receiveMessage(params, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  });
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [ReceiveMessage AWS SDK for JavaScriptAPI-Referenz](#).

SendMessage

Das folgende Codebeispiel zeigt die Verwendung `SendMessage`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie eine Nachricht an eine Amazon SQS SQS-Warteschlange.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
```

```
MessageAttributes: {
  Title: {
    DataType: "String",
    StringValue: "The Whistler",
  },
  Author: {
    DataType: "String",
    StringValue: "John Grisham",
  },
  WeeksOn: {
    DataType: "Number",
    StringValue: "6",
  },
},
MessageBody:
  "Information about current NY Times fiction bestseller for week of
12/11/2016.",
});

const response = await client.send(command);
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SendMessage](#) unter AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Senden Sie eine Nachricht an eine Amazon SQS SQS-Warteschlange.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler", // Required for FIFO queues
  // MessageGroupId: "Group1", // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SendMessage](#) unter AWS SDK for JavaScript API-Referenz.

SetQueueAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetQueueAttributes`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Konfigurieren Sie eine Amazon SQS SQS-Warteschlange für die Verwendung von Long Polling.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });
};
```

```
const response = await client.send(command);
console.log(response);
return response;
};
```

Konfigurieren Sie eine Warteschlange für unzustellbare Nachrichten.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
        SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie unter [SetQueueAttributes AWS SDK for JavaScript API-Referenz](#).

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Dies ist der Einstiegspunkt für diesen Workflow.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Der obige Code stellt die erforderlichen Abhängigkeiten bereit und startet den Workflow. Der nächste Abschnitt enthält den Großteil des Beispiels.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../..libs/prompter.js').Prompter} prompter
   * @param {import('../..libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }
}
```

```
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.snsClient.send(
    new CreateTopicCommand({
      Name: this.topicName,
      Attributes: {
        FifoTopic: this.isFifo ? "true" : "false",
        ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
      },
    }),
  );

  this.topicArn = response.TopicArn;
}
```

```
await this.logger.log(
  MESSAGES.topicCreatedNotice
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TOPIC_ARN}", this.topicArn),
);
}

async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  let maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });
    if (this.isFifo) {
      queueName += ".fifo";
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.sqsClient.send(
      new CreateQueueCommand({
        QueueName: queueName,
        Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
      }),
    );

    const { Attributes } = await this.sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: response.QueueUrl,
        AttributeNames: ["QueueArn"],
      }),
    );

    this.queues.push({
      queueName,
      queueArn: Attributes.QueueArn,
    });
  }
}
```

```
        queueUrl: response.QueueUrl,
    });

    await this.logger.log(
        MESSAGES.queueCreatedNotice
            .replace("${QUEUE_NAME}", queueName)
            .replace("${QUEUE_URL}", response.QueueUrl)
            .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
}
}

async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
        const policy = JSON.stringify(
            {
                Statement: [
                    {
                        Effect: "Allow",
                        Principal: {
                            Service: "sns.amazonaws.com",
                        },
                        Action: "sqs:SendMessage",
                        Resource: queue.queueArn,
                        Condition: {
                            ArnEquals: {
                                "aws:SourceArn": this.topicArn,
                            },
                        },
                    },
                ],
            },
            null,
            2,
        );

        if (index !== 0) {
            this.logger.logSeparator();
        }

        await this.logger.log(MESSAGES.attachPolicyNotice);
        console.log(policy);
        const addPolicy = await this.prompter.confirm({
            message: MESSAGES.addPolicyConfirmation.replace(
```

```
        "${QUEUE_NAME}",
        queue.queueName,
    ),
});

if (addPolicy) {
    await this.sqsClient.send(
        new SetQueueAttributesCommand({
            QueueUrl: queue.queueUrl,
            Attributes: {
                Policy: policy,
            },
        }),
    );
    queue.policy = policy;
} else {
    await this.logger.log(
        MESSAGES.policyNotAttachedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
        ),
    );
}
}
}

async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
        /**
         * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
         */
        const subscribeParams = {
            TopicArn: this.topicArn,
            Protocol: "sqs",
            Endpoint: queue.queueArn,
        };
        let tones = [];

        if (this.isFifo) {
            if (index === 0) {
                await this.logger.log(MESSAGES.fifoFilterNotice);
            }
            tones = await this.prompter.checkbox({
                message: MESSAGES.fifoFilterSelect.replace(
```

```
        "${QUEUE_NAME}",
        queue.queueName,
    ),
    choices: toneChoices,
});

if (tones.length) {
    subscribeParams.Attributes = {
        FilterPolicyScope: "MessageAttributes",
        FilterPolicy: JSON.stringify({
            tone: tones,
        }),
    };
}

const { SubscriptionArn } = await this.snsClient.send(
    new SubscribeCommand(subscribeParams),
);

this.subscriptionArns.push(SubscriptionArn);

await this.logger.log(
    MESSAGES.queueSubscribedNotice
        .replace("${QUEUE_NAME}", queue.queueName)
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
);
}
}

async publishMessages() {
    const message = await this.prompter.input({
        message: MESSAGES.publishMessagePrompt,
    });

    let groupId, deduplicationId, choices;

    if (this.isFifo) {
        await this.logger.log(MESSAGES.groupIdNotice);
        groupId = await this.prompter.input({
            message: MESSAGES.groupIdPrompt,
        });
    }
}
```

```
if (this.autoDedup === false) {
  await this.logger.log(MESSAGES.deduplicationIdNotice);
  deduplicationId = await this.prompter.input({
    message: MESSAGES.deduplicationIdPrompt,
  });
}

choices = await this.prompter.checkbox({
  message: MESSAGES.messageAttributesPrompt,
  choices: toneChoices,
});
}

await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
    ...(choices
      ? {
          MessageAttributes: {
            tone: {
              DataType: "String.Array",
              StringValue: JSON.stringify(choices),
            },
          },
        }
      : {}),
  })),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});
```



```
    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }

    const deleteAndPoll = await this.prompter.confirm({
      message: MESSAGES.deleteAndPollConfirmation,
    });
  }
}
```

```
    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
    }
  }
}
```

```
    this.logger.logSeparator(MESSAGES.headerReceiveMessages);
    await this.receiveAndDeleteMessages();
  } catch (err) {
    console.error(err);
  } finally {
    await this.destroyResources();
  }
}
}
```


- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. 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;
  }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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}`);
}

```

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von 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.');
```

```
}
```

Step Functions Functions-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie die Funktionen AWS SDK for JavaScript (v3) mit Step verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

StartExecution

Das folgende Codebeispiel zeigt, wie man es benutzt `StartExecution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
// SPDX-License-Identifier: Apache-2.0
```

```
import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-
east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

- Einzelheiten zur API finden Sie [StartExecution](#) in der AWS SDK for JavaScript API-Referenz.

AWS STS Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden AWS STS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- [Aktionen](#)

Aktionen

AssumeRole

Das folgende Codebeispiel zeigt, wie man es benutzt `AssumeRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { STSClient } from "@aws-sdk/client-sts";
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Übernehmen Sie die IAM-Rolle.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";
```

```
import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [AssumeRole](#) in der AWS SDK for JavaScript API-Referenz.

SDK für JavaScript (v2)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
// Load the AWS SDK for Node.js
const AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

var roleToAssume = {
  RoleArn: "arn:aws:iam::123456789012:role/RoleName",
  RoleSessionName: "session1",
```

```
    DurationSeconds: 900,
  };
  var roleCreds;

  // Create the STS service object
  var sts = new AWS.STS({ apiVersion: "2011-06-15" });

  //Assume Role
  sts.assumeRole(roleToAssume, function (err, data) {
    if (err) console.log(err, err.stack);
    else {
      roleCreds = {
        accessKeyId: data.Credentials.AccessKeyId,
        secretAccessKey: data.Credentials.SecretAccessKey,
        sessionToken: data.Credentials.SessionToken,
      };
      stsGetCallerIdentity(roleCreds);
    }
  });

  //Get Arn of current identity
  function stsGetCallerIdentity(creds) {
    var stsParams = { credentials: creds };
    // Create STS service object
    var sts = new AWS.STS(stsParams);

    sts.getCallerIdentity({}, function (err, data) {
      if (err) {
        console.log(err, err.stack);
      } else {
        console.log(data.Arn);
      }
    });
  }
}
```

- Einzelheiten zur API finden Sie [AssumeRole](#) in der AWS SDK for JavaScript API-Referenz.

AWS Support Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK for JavaScript (v3) mit verwenden AWS Support.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Support

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Support beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Rufen Sie „main()“ auf, um das Beispiel auszuführen.

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });

const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
```

```
    throw new Error(
      "You must be subscribed to the AWS Support plan to use this feature.",
    );
  } else {
    throw err;
  }
}
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeServices](#) in der AWS SDK for JavaScript API-Referenz.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AddAttachmentsToSet

Das folgende Codebeispiel zeigt die Verwendung `AddAttachmentsToSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";
```

```
import { client } from "../libs/client.js";


export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
    a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
        attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [AddAttachmentsToSet](#) in der AWS SDK for JavaScript API-Referenz.

AddCommunicationToCase

Das folgende Codebeispiel zeigt die Verwendung `AddCommunicationToCase`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
        // to the case.
        attachmentSetId,
      }),
    );
    console.log(response);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [AddCommunicationToCase](#) in der AWS SDK for JavaScript API-Referenz.

CreateCase

Das folgende Codebeispiel zeigt die Verwendung `CreateCase`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { CreateCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new case and log the case id.
    // Important: This creates a real support case in your account.
    const response = await client.send(
      new CreateCaseCommand({
        // The subject line of the case.
        subject: "IGNORE: Test case",
        // Use DescribeServices to find available service codes for each service.
        serviceCode: "service-quicksight-end-user",
        // Use DescribeSecurityLevels to find available severity codes for your
        support plan.
        severityCode: "low",
        // Use DescribeServices to find available category codes for each service.
        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
      }),
    );
    console.log(response.caseId);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [CreateCase](#) in der AWS SDK for JavaScript API-Referenz.

DescribeAttachment

Das folgende Codebeispiel zeigt die Verwendung `DescribeAttachment`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";


export const main = async () => {
  try {
    // Get the metadata and content of an attachment.
    const response = await client.send(
      new DescribeAttachmentCommand({
        // Set value to an existing attachment id.
        // Use DescribeCommunications or DescribeCases to find an attachment id.
        attachmentId: "ATTACHMENT_ID",
      }),
    );
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeAttachment](#) in der AWS SDK for JavaScript API-Referenz.

DescribeCases

Das folgende Codebeispiel zeigt die Verwendung `DescribeCases`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";


export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
    Refer
    // to the TypeScript definition and the API doc for more information on possible
    parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
    support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeCases](#) in der AWS SDK for JavaScript API-Referenz.

DescribeCommunications

Das folgende Codebeispiel zeigt die Verwendung `DescribeCommunications`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all communications for the support case.
    // Filter results by providing parameters to the DescribeCommunicationsCommand.
    Refer
    // to the TypeScript definition and the API doc for more information on possible
    parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
    support/interfaces/describecommunicationscommandinput.html
    const response = await client.send(
      new DescribeCommunicationsCommand({
        // Set value to an existing case id.
        caseId: "CASE_ID",
      }),
    );
    const text = response.communications.map((item) => item.body).join("\n");
    console.log(text);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeCommunications](#) in der AWS SDK for JavaScript API-Referenz.

DescribeSeverityLevels

Das folgende Codebeispiel zeigt die Verwendung `DescribeSeverityLevels`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get the list of severity levels.
    // The available values depend on the support plan for the account.
    const response = await client.send(new DescribeSeverityLevelsCommand({}));
    console.log(response.severityLevels);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeSeverityLevels](#) in der AWS SDK for JavaScript API-Referenz.

ResolveCase

Das folgende Codebeispiel zeigt die Verwendung `ResolveCase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [ResolveCase](#) in der AWS SDK for JavaScript API-Referenz.

Szenarien

Erste Schritte mit Fällen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie verfügbare Services und Schweregrade für Fälle ab und zeigen Sie sie an.
- Erstellen Sie einen Supportfall mit einem ausgewählten Service, einer ausgewählten Kategorie und einem ausgewählten Schweregrad.
- Rufen Sie eine Liste der offenen Fälle für den aktuellen Tag ab und zeigen Sie sie an.

- Fügen Sie dem neuen Fall einen Anhangssatz und eine Mitteilung hinzu.
- Beschreiben Sie den neuen Anhang und die Mitteilung für den Fall.
- Lösen Sie den Fall.
- Rufen Sie eine Liste der gelösten Fälle für den aktuellen Tag ab und zeigen Sie sie an.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario im Terminal aus.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n  ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});
```

```
    try {
      await client.send(command);
    } catch (err) {
      if (err.name === "SubscriptionRequiredException") {
        throw new Error(
          "You must be subscribed to the AWS Support plan to use this feature.",
        );
      } else {
        throw err;
      }
    }
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[] }} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
  });
};
```

```
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
 * @param {{
 *   selectedService: import('@aws-sdk/client-support').Service
 *   selectedCategory: import('@aws-sdk/client-support').Category
 *   selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
 * }} selections
 * @returns
 */
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodaysOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
};
```



```
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
};
```

```
    return firstCommWithAttachment?.attachmentSet[0].attachmentId;
  };

  // Get an attachment.
  export const getAttachment = async (attachmentId) => {
    const command = new DescribeAttachmentCommand({
      attachmentId,
    });
    const { attachment } = await client.send(command);
    return attachment;
  };

  // Resolve the case matching the given case ID.
  export const resolveCase = async (caseId) => {
    const shouldResolve = await inquirer.confirm({
      message: `Do you want to resolve ${caseId}?`,
    });

    if (shouldResolve) {
      const command = new ResolveCaseCommand({
        caseId: caseId,
      });

      await client.send(command);
      return true;
    }
    return false;
  };

  /**
   * Find a specific case in the list of provided cases by case ID.
   * If the case is not found, and the results are paginated, continue
   * paging through the results.
   * @param {{
   *   caseId: string,
   *   cases: import('@aws-sdk/client-support').CaseDetails[]
   *   nextToken: string
   * }} options
   * @returns
   */
  export const findCase = async ({ caseId, cases, nextToken }) => {
    const foundCase = cases.find((c) => c.caseId === caseId);

    if (foundCase) {
```

```
    return foundCase;
  }

  if (nextToken) {
    const response = await client.send(
      new DescribeCasesCommand({
        nextToken,
        includeResolvedCases: true,
      }),
    );
    return findCase({
      caseId,
      cases: response.cases,
      nextToken: response.nextToken,
    });
  }

  throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();
```

```
// Provided the categories for the selected service and prompt the user to
select one.
const selectedCategory = await getCategory(selectedService);

// Provide the severity available severity levels for the account and prompt the
user to select one.
const selectedSeverityLevel = await getSeverityLevel();

// Create a support case.
console.log("\nCreating a support case.");
caseId = await createCase({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
});
console.log(`Support case created: ${caseId}`);

// Display a list of open support cases created today.
const todaysOpenCases = await retry(
  { intervalInMs: 1000, maxRetries: 15 },
  getTodaysOpenCases,
);
console.log(
  "\nOpen support cases created today: ${todaysOpenCases.length}`,
);
console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

// Create an attachment set.
console.log("\nCreating an attachment set.");
const attachmentSetId = await createAttachmentSet();
console.log(`Attachment set created: ${attachmentSetId}`);

// Add the attachment set to the support case.
console.log(`\nAdding attachment set to ${caseId}`);
await linkAttachmentSetToCase(attachmentSetId, caseId);
console.log(`Attachment set added to ${caseId}`);

// List the communications for a support case.
console.log(`\nListing communications for ${caseId}`);
const communications = await getCommunications(caseId);
console.log(
  communications
  .map(
```

```
        (c) =>
            `Communication created on ${c.timeCreated}. Has
            ${c.attachmentSet.length} attachments.` ,
        )
        .join("\n"),
    );

// Describe the first attachment.
console.log(`\nDescribing attachment ${attachmentSetId}`);
const attachmentId = getFirstAttachment(communications);
const attachment = await getAttachment(attachmentId);
console.log(
    `Attachment is the file '${
        attachment.fileName
    }' with data: \n${new TextDecoder().decode(attachment.data)}`,
);

// Confirm that the support case should be resolved.
const isResolved = await resolveCase(caseId);
if (isResolved) {
    // List the resolved cases and include the one previously created.
    // Resolved cases can take a while to appear.
    console.log(
        "\nWaiting for case status to be marked as resolved. This can take some
time.",
    );
    const resolvedCases = await retry(
        { intervalInMs: 20000, maxRetries: 15 },
        () => getTodaysResolvedCases(caseId),
    );
    console.log("Resolved cases:");
    console.log(resolvedCases.map((c) => c.caseId).join("\n"));
}
} catch (err) {
    console.error(err);
}
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK for JavaScript -API-Referenz.
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)

- [CreateCase](#)
- [DescribeAttachment](#)
- [DescribeCases](#)
- [DescribeCommunications](#)
- [DescribeServices](#)
- [DescribeSeverityLevels](#)
- [ResolveCase](#)

Amazon Transcribe Transcribe-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK for JavaScript (v3) mit Amazon Transcribe Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Servicefunktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien und serviceübergreifenden Beispiele sehen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie eine bestimmte Aufgabe ausführen können, indem Sie mehrere Funktionen innerhalb desselben Services aufrufen.

Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen finden, wie Sie den Code im Kontext einrichten und ausführen können.

Themen


- [Aktionen](#)

Aktionen

DeleteMedicalTranscriptionJob

Das folgende Codebeispiel zeigt, wie man es benutzt `DeleteMedicalTranscriptionJob`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Einen Auftrag für medizinische Transkription löschen.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteMedicalTranscriptionJob](#) in der AWS SDK for JavaScript API-Referenz.

DeleteTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `DeleteTranscriptionJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Löschen eines Transkriptionsauftrags.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params)
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```



```
run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteTranscriptionJob](#) in der AWS SDK for JavaScript API-Referenz.

ListMedicalTranscriptionJobs

Das folgende Codebeispiel zeigt die Verwendung `ListMedicalTranscriptionJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

Auflisten medizinischer Transkriptionsjobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    // region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListMedicalTranscriptionJobs](#) in der AWS SDK for JavaScript API-Referenz.

ListTranscriptionJobs

Das folgende Codebeispiel zeigt die Verwendung `ListTranscriptionJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Auflisten von Transkriptionsaufträgen.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params)
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTranscriptionJobs](#) in der AWS SDK for JavaScript API-Referenz.

StartMedicalTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `StartMedicalTranscriptionJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Einen Auftrag für medizinische Transkription starten.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
```

```
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [StartMedicalTranscriptionJob](#) in der AWS SDK for JavaScript API-Referenz.

StartTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `StartTranscriptionJob`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Sie sehen das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-Repository](#) einrichten und ausführen.

Einen Transkriptionsauftrag starten.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME"
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params)
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK for JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [StartTranscriptionJob](#) in der AWS SDK for JavaScript API-Referenz.

Serviceübergreifende Beispiele mit SDK für JavaScript (v3)

Die folgenden Beispielanwendungen verwenden die AWS SDK for JavaScript (v3), um mit mehreren zu arbeiten AWS-Services.

Serviceübergreifende Beispiele zielen auf fortgeschrittene Erfahrung ab, damit Sie mit der Erstellung von Anwendungen beginnen können.

Beispiele

- [Eine Amazon-Transcribe-App entwickeln](#)
- [Erstellen einer Amazon-Transcribe-Streaming-App](#)
- [Erstellen Sie eine Anwendung zum Senden von Daten an eine DynamoDB-Tabelle](#)
- [Erstellen Sie einen Amazon Lex Lex-Chatbot, um die Besucher Ihrer Website anzusprechen](#)
- [Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können](#)
- [Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten](#)
- [Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben](#)
- [Erstellen Sie eine Amazon-Textract-Explorer-Anwendung](#)
- [Erstellen einer Anwendung, die Kundenfeedback analysiert und Audio generiert](#)
- [Ermitteln Sie persönliche Schutzausrüstung in Bildern mit Amazon Rekognition mithilfe eines SDK AWS](#)
- [Objekte in Bildern mit Amazon Rekognition mithilfe eines SDK erkennen AWS](#)

- [Erkennen Sie Personen und Objekte in einem Video mit Amazon Rekognition mithilfe eines SDK AWS](#)
- [Aufrufen einer Lambda-Funktion von einem Browser aus](#)
- [Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion](#)
- [Verwenden von Step Functions, um Lambda-Funktionen aufzurufen](#)
- [Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion](#)

Eine Amazon-Transcribe-App entwickeln

SDK für JavaScript (v3)

Erstellen Sie eine App, die Amazon Transcribe verwendet, um Sprachaufnahmen im Browser zu transkribieren und anzuzeigen. Die App verwendet zwei Amazon Simple Storage Service (Amazon S3)-Buckets, einen zum Hosten des Anwendungscodes und einen zum Speichern von Transkriptionen. Die App verwendet einen Amazon-Cognito-Benutzerpool zur Authentifizierung Ihrer Benutzer. Authentifizierte Benutzer verfügen über AWS Identity and Access Management (IAM-) Berechtigungen für den Zugriff auf die erforderlichen AWS Dienste.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

Dieses Beispiel ist auch verfügbar im [AWS SDK for JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon Transcribe

Erstellen einer Amazon-Transcribe-Streaming-App

SDK für JavaScript (v3)

Zeigt, wie Amazon Transcribe verwendet wird, um eine App zu erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse mit Amazon Simple Email Service (Amazon SES) per E-Mail sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Erstellen Sie eine Anwendung zum Senden von Daten an eine DynamoDB-Tabelle

SDK für JavaScript (v3)

Das Beispiel zeigt, wie man eine App erstellt, die es Benutzern ermöglicht, Daten an eine Amazon-DynamoDB-Tabelle zu übermitteln und eine Textnachricht an den Administrator mit Amazon Simple Notification Service (Amazon SNS) zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK for JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SNS

Erstellen Sie einen Amazon Lex Lex-Chatbot, um die Besucher Ihrer Website anzusprechen

SDK für JavaScript (v3)

Zeigt, wie Sie mithilfe der Amazon Lex Lex-API einen Chatbot innerhalb einer Webanwendung erstellen, um die Besucher Ihrer Website anzusprechen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel „[Einen Amazon Lex-Chatbot erstellen](#)“ im AWS SDK for JavaScript Entwicklerhandbuch.

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Eine Anwendung für Foto-Asset-Management erstellen, mit der Benutzer Fotos mithilfe von Labels verwalten können

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erstellen einer Webanwendung zur Verfolgung von DynamoDB-Daten

SDK für JavaScript (v3)

Zeigt, wie man die Amazon-DynamoDB-API verwendet, um eine dynamische Webanwendung zu erstellen, die DynamoDB-Arbeitsdaten verfolgt.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SES

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK for JavaScript (v3) eine Webanwendung erstellen, die Arbeitsaufgaben in einer Amazon Aurora Aurora-Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service

- Amazon SES

Erstellen Sie eine Amazon-Textextract-Explorer-Anwendung

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK for JavaScript dem eine React-Anwendung erstellen, die Amazon Textract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Erstellen einer Anwendung, die Kundenfeedback analysiert und Audio generiert

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.

- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Den Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK for JavaScript wird.

```
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 };
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ermitteln Sie persönliche Schutzausrüstung in Bildern mit Amazon Rekognition mithilfe eines SDK AWS

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition mit dem verwendet wird, AWS SDK for JavaScript um eine Anwendung zur Erkennung persönlicher Schutzausrüstung (PSA) in Bildern zu erstellen, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App speichert die Ergebnisse in einer Amazon-DynamoDB-Tabelle und sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

So funktioniert es:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für PSA.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Aktualisieren Sie eine DynamoDB-Tabelle mit Ergebnissen.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Objekte in Bildern mit Amazon Rekognition mithilfe eines SDK erkennen AWS

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition zusammen mit dem verwendet wird, um eine App AWS SDK for JavaScript zu erstellen, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu identifizieren, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

So funktioniert es:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für Objekte.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES

Erkennen Sie Personen und Objekte in einem Video mit Amazon Rekognition mithilfe eines SDK AWS

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition mit dem verwendet wird, AWS SDK for JavaScript um eine App zur Erkennung von Gesichtern und Objekten in Videos zu erstellen, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

So funktioniert es:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für PSA.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES

Aufrufen einer Lambda-Funktion von einem Browser aus

SDK für JavaScript (v2)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#)

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda

SDK für JavaScript (v3)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren. Diese App verwendet v3. AWS SDK for JavaScript

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK for JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- API Gateway

- DynamoDB
- Lambda
- Amazon SNS

Verwenden von Step Functions, um Lambda-Funktionen aufzurufen

SDK für JavaScript (v3)

Zeigt, wie Sie einen AWS serverlosen Workflow mithilfe von AWS Step Functions erstellen. AWS SDK for JavaScript Jeder Workflow-Schritt wird mithilfe einer AWS Lambda Funktion implementiert.

Lambda ist ein Datenverarbeitungsservice, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Step Functions ist ein Serverless-Orchestrierungsservice, mit dem Sie Lambda-Funktionen und andere kombinieren AWS -Services kombinieren können, um geschäftskritische Anwendungen zu erstellen.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK for JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen.

Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK for JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Sicherheit für dieses AWS Produkt oder diese Dienstleistung

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS -Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Im [Modell der übergreifenden Verantwortlichkeit](#) wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der [AWS Compliance-Programme](#) regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Themen

- [Datenschutz in diesem AWS Produkt oder dieser Dienstleistung](#)
- [Identitäts- und Zugriffsverwaltung](#)
- [Konformitätsprüfung für dieses Produkt oder diese Dienstleistung AWS](#)
- [Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung](#)
- [Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service](#)
- [Erzwingen Sie eine TLS-Mindestversion](#)

Datenschutz in diesem AWS Produkt oder dieser Dienstleistung

Das AWS [Modell](#) der gilt für den Datenschutz in diesem AWS Produkt oder dieser Dienstleistung. Wie in diesem Modell beschrieben, AWS ist es verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich.

Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#). Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag [AWS -Modell der geteilten Verantwortung und in der DSGVO](#) im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-2](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit diesem AWS Produkt oder Service oder einem anderen AWS-Services über die Konsole, API oder SDKs arbeiten. AWS CLI AWS Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine

Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Identitäts- und Zugriffsverwaltung

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service, den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IAM ist ein Programm AWS-Service, das Sie ohne zusätzliche Kosten nutzen können.

Themen

- [Zielgruppe](#)
- [Authentifizierung mit Identitäten](#)
- [Verwalten des Zugriffs mit Richtlinien](#)
- [Wie AWS-Services arbeiten Sie mit IAM](#)
- [Fehlerbehebung bei AWS Identität und Zugriff](#)

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS

Dienstbenutzer — Wenn Sie dies AWS-Services für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Falls Sie auf eine Funktion nicht zugreifen können AWS, finden [Fehlerbehebung bei AWS Identität und Zugriff](#) Sie weitere Informationen in der Bedienungsanleitung der von AWS-Service Ihnen verwendeten.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen.

Weitere Informationen darüber, wie Ihr Unternehmen IAM verwenden kann AWS, finden Sie in der Benutzeranleitung des von AWS-Service Ihnen verwendeten.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS verfassen können. Beispiele für AWS identitätsbasierte Richtlinien, die Sie in IAM verwenden können, finden Sie im Benutzerhandbuch der AWS-Service von Ihnen verwendeten.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS, übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangportal anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter [So melden Sie sich bei Ihrem an AWS-Konto](#) im AWS-Anmeldung Benutzerhandbuch.

Wenn Sie AWS programmgesteuert zugreifen, AWS stellt es ein Software Development Kit (SDK) und eine Befehlszeilenschnittstelle (CLI) bereit, um Ihre Anfragen mithilfe Ihrer Anmeldeinformationen kryptografisch zu signieren. Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode, um Anfragen selbst zu [signieren, finden Sie im IAM-Benutzerhandbuch unter AWS API-Anfragen](#) signieren.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter [Multi-Faktor-Authentifizierung](#) im AWS IAM Identity Center - Benutzerhandbuch und [Verwenden der Multi-Faktor-Authentifizierung \(MFA\) in AWS](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie ein AWS-Konto erstellen, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen den Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, einem Web-Identitätsanbieter AWS Directory Service, dem Identity Center-Verzeichnis oder einem beliebigen Benutzer, der mithilfe AWS-Services von Anmeldeinformationen zugreift, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität innerhalb Ihres Unternehmens AWS-Konto, die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter [Regelmäßiges](#)

[Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Eine [IAM-Gruppe](#) ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter [Erstellen eines IAM-Benutzers \(anstatt einer Rolle\)](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität innerhalb Ihres Unternehmens AWS-Konto, die über bestimmte Berechtigungen verfügt. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS Management Console indem Sie die Rollen [wechseln](#). Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter [Verwenden von IAM-Rollen](#) im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- **Verbundbenutzerzugriff** – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter [Erstellen von Rollen für externe Identitätsanbieter](#) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter [Berechtigungssätze](#) im AWS IAM Identity Center -Benutzerhandbuch.
- **Temporäre IAM-Benutzerberechtigungen** – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.

- **Kontoübergreifender Zugriff** – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zum Unterschied zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch](#).
- **Serviceübergreifender Zugriff** — Einige verwenden Funktionen in anderen. AWS-Services AWS-Services Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon-EC2 aus oder speichert Objekte in Amazon-S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
- **Forward Access Sessions (FAS)** — Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle verwenden, um Aktionen auszuführen AWS, gelten Sie als Principal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service initiieren. FAS verwendet die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anfrage, Anfragen an AWS-Service nachgelagerte Dienste zu stellen. FAS-Anfragen werden nur gestellt, wenn ein Dienst eine Anfrage erhält, für deren Abschluss Interaktionen mit anderen AWS-Services oder Ressourcen erforderlich sind. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter [Zugriffssitzungen weiterleiten](#).
- **Servicerolle** – Eine Servicerolle ist eine [IAM-Rolle](#), die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service](#) im IAM-Benutzerhandbuch.
- **Dienstbezogene Rolle** — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.
- **Auf Amazon EC2 ausgeführte Anwendungen** — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und API-Anfragen stellen AWS CLI . AWS Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Um einer EC2-Instance eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie

ein Instance-Profil, das an die Instance angehängt ist. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter [Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon-EC2-Instances ausgeführt werden](#) im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter [Erstellen einer IAM-Rolle \(anstatt eines Benutzers\)](#) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anfrage stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS Management Console AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern,

welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Erstellen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem System zuordnen können AWS-Konto. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter [Auswahl zwischen verwalteten und eingebundenen Richtlinien](#) im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#). Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Services, die ACLs unterstützen. AWS WAF Weitere Informationen“ zu ACLs finden Sie unter [Zugriffskontrollliste \(ACL\) – Übersicht](#) (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- **Berechtigungsgrenzen** – Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im IAM-Benutzerhandbuch.
- **Service Control Policies (SCPs)** — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) in festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer Objekte AWS-Konten , die Ihrem Unternehmen gehören. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Entitäten. Root-Benutzer des AWS-Kontos Weitere Informationen zu Organizations und SCPs finden Sie unter [Funktionsweise von SCPs](#) im AWS Organizations -Benutzerhandbuch.
- **Sitzungsrichtlinien** – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter [Sitzungsrichtlinien](#) im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird,

ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie im IAM-Benutzerhandbuch unter [Bewertungslogik für Richtlinien](#).

Wie AWS-Services arbeiten Sie mit IAM

Einen allgemeinen Überblick darüber, wie die meisten IAM-Funktionen AWS-Services funktionieren, finden Sie im [AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren](#).

Informationen zur Verwendung bestimmter Dienste AWS-Service mit IAM finden Sie im Abschnitt Sicherheit im Benutzerhandbuch des jeweiligen Dienstes.

Fehlerbehebung bei AWS Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS und IAM auftreten können.

Themen

- [Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS](#)
- [Ich bin nicht berechtigt, iam auszuführen: PassRole](#)
- [Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen](#)

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven `my-example-widget`-Ressource anzuzeigen, jedoch nicht über `aws:GetWidget`-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `aws:GetWidget`-Aktion auf die `my-example-widget`-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Im Fall von Diensten, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS unterstützt werden, finden Sie unter [Wie AWS-Services arbeiten Sie mit IAM](#).
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter [Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto, den Sie besitzen](#).

- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie [AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte](#).
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter [Gewähren von Zugriff für extern authentifizierte Benutzer \(Identitätsverbund\)](#) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie im IAM-Benutzerhandbuch unter [Kontenübergreifender Ressourcenzugriff in IAM](#).

Konformitätsprüfung für dieses Produkt oder diese Dienstleistung AWS

Informationen darüber, ob AWS-Service ein [AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt](#), finden Sie unter [Umfang nach Compliance-Programm AWS-Services unter](#) . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme AWS](#) .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter [Berichte herunterladen unter](#) .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- [Schnellstartanleitungen zu Sicherheit und Compliance](#) — In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Bereitstellung von Basisumgebungen beschrieben AWS , bei denen Sicherheit und Compliance im Mittelpunkt stehen.
- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) — In diesem Whitepaper wird beschrieben, wie Unternehmen HIPAA-fähige Anwendungen erstellen AWS können.

Note

AWS-Services Nicht alle sind HIPAA-fähig. Weitere Informationen finden Sie in der [Referenz für HIPAA-berechtigte Services](#).

- [AWS Compliance-Ressourcen](#) — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- [AWS Leitfäden zur Einhaltung von Vorschriften für Kunden](#) — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS-Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- [Evaluierung von Ressourcen anhand von Regeln](#) im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#)— Auf diese AWS-Service Weise erhalten Sie einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Eine Liste der unterstützten Services und Kontrollen finden Sie in der [Security-Hub-Steuerungsreferenz](#).
- [Amazon GuardDuty](#) — Dies AWS-Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.
- [AWS Audit Manager](#)— Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones.

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind.

Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Dienst verwendet Managed Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf dieses AWS Produkt oder diesen Service zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit [AWS](#)

[Security Token Service](#) (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Erzwingen Sie eine TLS-Mindestversion

Um die Sicherheit bei der Kommunikation mit AWS Diensten zu erhöhen, konfigurieren Sie den AWS SDK for JavaScript für die Verwendung von TLS 1.2 oder höher.

Important

Die Version AWS SDK for JavaScript 3 handelt automatisch die höchste TLS-Version aus, die von einem bestimmten AWS Dienstendpunkt unterstützt wird. Sie können optional eine TLS-Mindestversion erzwingen, die für Ihre Anwendung erforderlich ist, z. B. TLS 1.2 oder 1.3. Beachten Sie jedoch, dass TLS 1.3 von einigen AWS Service-Endpunkten nicht unterstützt wird, sodass einige Aufrufe fehlschlagen können, wenn Sie TLS 1.3 erzwingen.

Transport Layer Security (TLS) ist ein Protokoll, das von Webbrowsern und anderen Anwendungen verwendet wird, um die Privatsphäre und Integrität der über ein Netzwerk ausgetauschten Daten zu gewährleisten.

Überprüfen und Erzwingen von TLS in Node.js

Wenn Sie das AWS SDK for JavaScript mit Node.js verwenden, wird die zugrunde liegende Sicherheitsebene Node.js verwendet, um die TLS-Version festzulegen.

Node.js 12.0.0 und höher verwenden eine Mindestversion von OpenSSL 1.1.1b, die TLS 1.3 unterstützt. AWS SDK for JavaScript Version 3 verwendet standardmäßig TLS 1.3, sofern verfügbar, verwendet jedoch standardmäßig eine niedrigere Version, falls erforderlich.

Überprüfen der Version von OpenSSL und TLS

Führen Sie den folgenden Befehl aus, um die von Node.js verwendete Version von OpenSSL auf Ihrem Computer abzurufen.

```
node -p process.versions
```

Die Version von OpenSSL in der Liste ist die von Node.js verwendete Version, wie im folgenden Beispiel gezeigt.

```
openssl: '1.1.1b'
```

Um die von Node.js verwendete Version von TLS auf Ihrem Computer abzurufen, starten Sie die Knoten-Shell und führen Sie die folgenden Befehle in der angegebenen Reihenfolge aus.

```
> var tls = require("tls");  
> var tlsSocket = new tls.TLSSocket();  
> tlsSocket.getProtocol();
```

Der letzte Befehl gibt die TLS-Version aus, wie im folgenden Beispiel gezeigt.

```
'TLSv1.3'
```

Node.js verwendet standardmäßig diese Version von TLS und versucht, eine andere Version von TLS auszuhandeln, wenn ein Aufruf nicht erfolgreich ist.

Erzwingen einer Mindestversion von TLS

Node.js verhandelt eine Version von TLS, wenn ein Aufruf fehlschlägt. Sie können während dieser Verhandlung die zulässige Mindestversion von TLS durchsetzen, entweder wenn Sie ein Skript über die Befehlszeile ausführen oder per Anfrage in Ihrem Code. JavaScript

Um die minimale TLS-Version über die Befehlszeile anzugeben, müssen Sie Node.js Version 11.0.0 oder höher verwenden. Um eine bestimmte Version von Node.js zu installieren, installieren Sie zunächst Node Version Manager (nvm). Gehen Sie dabei wie unter [Installation und Aktualisierung des Node Version Managers beschrieben](#) vor. Führen Sie dann die folgenden Befehle aus, um eine bestimmte Version von Node.js zu installieren und zu verwenden.

```
nvm install 11  
nvm use 11
```

Enforce TLS 1.2

Um zu erzwingen, dass TLS 1.2 die minimal zulässige Version ist, geben Sie beim Ausführen des Skripts das Argument `--tls-min-v1.2` an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.2 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `httpOptions` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  region: "us-west-2",
  requestHandler: new NodeHttpHandler({
    httpsAgent: new https.Agent({
      {
        secureProtocol: 'TLSv1_2_method'
      }
    })
  })
});
```

Enforce TLS 1.3

Um zu erzwingen, dass TLS 1.3 die zulässige Mindestversion ist, geben Sie das `--tls-min-v1.3` Argument bei der Ausführung Ihres Skripts an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.3 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `httpOptions` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
const client = new DynamoDBClient({
  region: "us-west-2",
  requestHandler: new NodeHttpHandler({
    httpsAgent: new https.Agent(
      {
        secureProtocol: 'TLSv1_3_method'
      }
    )
  })
});
```

Überprüfen und Erzwingen von TLS in einem Browserskript

Wenn Sie das SDK für JavaScript in einem Browserskript verwenden, steuern die Browsereinstellungen die verwendete TLS-Version. Die vom Browser verwendete Version von TLS kann nicht vom Skript erkannt oder festgelegt werden und muss vom Benutzer konfiguriert werden. Informationen zur Überprüfung und Erzwingung der in einem Browserskript verwendeten Version von TLS finden Sie in den Anweisungen für den jeweiligen Browser.

Microsoft Internet Explorer

1. Öffnen Sie Internet Explorer.
2. Wählen Sie in der Menüleiste die Option Extras — Internetoptionen — Registerkarte Erweitert.
3. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
4. Klicken Sie auf OK.
5. Schließen Sie Ihren Browser und starten Sie Internet Explorer neu.

Microsoft Edge

1. Geben Sie im Suchfeld des Windows-Menüs *Internetoptionen* ein.
2. Klicken Sie unter Beste Übereinstimmung auf Internetoptionen.
3. Scrollen Sie im Fenster Inteneteigenschaften auf der Registerkarte Erweitert nach unten zum Abschnitt Sicherheit.
4. Markieren Sie das Kontrollkästchen User TLS 1.2.

5. Klicken Sie auf OK.

Google Chrome

1. Öffnen Sie Google Chrome.
2. Klicken Sie auf Alt F und wählen Sie Einstellungen.
3. Scrollen Sie nach unten und wählen Sie Erweiterte Einstellungen anzeigen... .
4. Scrollen Sie nach unten zum Abschnitt System und klicken Sie auf Proxyeinstellungen öffnen... .
5. Wählen Sie die Registerkarte Erweitert aus.
6. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
7. Klicken Sie auf OK.
8. Schließen Sie Ihren Browser und starten Sie Google Chrome neu.

Mozilla Firefox

1. Öffnen Sie Firefox.
2. Geben Sie in der Adressleiste `about:config` ein und drücken Sie die Eingabetaste.
3. Geben Sie im Suchfeld `tls` ein. Suchen Sie den Eintrag für `security.tls.version.min` und doppelklicken Sie darauf.
4. Setzen Sie den Integer-Wert auf 3, um zu erzwingen, dass das Protokoll von TLS 1.2 als Standard verwendet wird.
5. Klicken Sie auf OK.
6. Schließen Sie Ihren Browser und starten Sie Mozilla Firefox neu.

Apple Safari

Es gibt keine Optionen zum Aktivieren von SSL-Protokollen. Wenn Sie Safari Version 7 oder höher verwenden, wird TLS 1.2 automatisch aktiviert.

Migrieren Sie von Version 2.x auf 3.x von AWS SDK for JavaScript

Die AWS SDK for JavaScript Version 3 ist eine umfassende Neufassung von Version 2. Der Abschnitt beschreibt die Unterschiede zwischen den beiden Versionen und erklärt, wie Sie von Version 2 auf Version 3 des SDK für JavaScript migrieren.

Migrieren Sie Ihren Code mithilfe von Codemod auf SDK für JavaScript v3

AWS SDK for JavaScript Version 3 (v3) bietet modernisierte Schnittstellen für Client-Konfigurationen und Dienstprogramme, darunter Anmeldeinformationen, mehrteiliges Hochladen von Amazon S3, DynamoDB-Dokumentenclient, Kellner und mehr. [Was sich in Version 2 geändert hat, und was in Version 3 für jede Änderung geändert wurde, finden Sie im Migrationsleitfaden im Repo. AWS SDK for JavaScript GitHub](#)

Um die Vorteile der Version 3 voll auszuschöpfen, empfehlen AWS SDK for JavaScript wir die Verwendung der unten beschriebenen Codemod-Skripte.

Verwenden Sie Codemod, um vorhandenen v2-Code zu migrieren

Die Sammlung von Codemod-Skripten in [aws-sdk-js-codemod](#) hilft Ihnen bei der Migration Ihrer vorhandenen AWS SDK for JavaScript (v2) -Anwendung zur Verwendung von v3-APIs. Sie können die Transformation wie folgt ausführen.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Stellen Sie sich zum Beispiel vor, Sie haben den folgenden Code, der einen Amazon DynamoDB-Client aus Version 2 erstellt und Operation aufruft `listTables`.

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise()
```

```
.then(console.log)
.catch(console.error);
```

Sie können unsere v2-to-v3 Transformation `example.ts` wie folgt ausführen.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

Die Transformation konvertiert den DynamoDB-Import in Version 3, erstellt einen v3-Client und ruft den `listTables` Vorgang wie folgt auf.

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
await client.listTables()
  .then(console.log)
  .catch(console.error);
```

Wir haben Transformationen für gängige Anwendungsfälle implementiert. Wenn Ihr Code nicht korrekt transformiert wird, erstellen Sie bitte einen [Fehlerbericht](#) oder eine [Funktionsanfrage](#) mit Beispiel-Eingabecode und beobachtetem/erwartetem Ausgabecode. Wenn Ihr spezieller Anwendungsfall bereits in [einem bestehenden Problem](#) gemeldet wurde, zeigen Sie Ihre Unterstützung durch eine positive Bewertung.

Was ist neu in Version 3

Version 3 des SDK für JavaScript (v3) enthält die folgenden neuen Funktionen.

Modularisierte Pakete

Benutzer können jetzt für jeden Dienst ein separates Paket verwenden.

Neuer Middleware-Stack

Benutzer können jetzt einen Middleware-Stack verwenden, um den Lebenszyklus eines Operationsaufrufs zu steuern.

Darüber hinaus ist das SDK integriert, was viele Vorteile bietet TypeScript, wie z. B. die statische Typisierung.

⚠ Important

Die Codebeispiele für Version 3 in diesem Handbuch sind in ECMAScript 6 (ES6) geschrieben. ES6 bietet neue Syntax und neue Funktionen, um Ihren Code moderner und lesbarer zu machen und mehr zu erreichen. ES6 erfordert, dass Sie Node.js Version 13.x oder höher verwenden. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js](#) downloads. Weitere Informationen finden Sie unter [JavaScript ES6/CommonJS-Syntax](#).

Modularisierte Pakete

Für Version 2 des SDK für JavaScript (v2) mussten Sie das gesamte AWS SDK wie folgt verwenden.

```
var AWS = require("aws-sdk");
```

Das Laden des gesamten SDK ist kein Problem, wenn Ihre Anwendung viele AWS Dienste verwendet. Wenn Sie jedoch nur wenige AWS Dienste verwenden müssen, bedeutet dies, dass Sie die Größe Ihrer Anwendung mit Code erhöhen müssen, den Sie nicht benötigen oder verwenden.

In Version 3 können Sie nur die einzelnen AWS Dienste laden und verwenden, die Sie benötigen. Dies wird im folgenden Beispiel gezeigt, das Ihnen Zugriff auf Amazon DynamoDB (DynamoDB) gewährt.

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

Sie können nicht nur einzelne AWS Dienste laden und verwenden, sondern Sie können auch nur die Servicebefehle laden und verwenden, die Sie benötigen. Dies wird in den folgenden Beispielen veranschaulicht, die Ihnen Zugriff auf den DynamoDB-Client und den `ListTablesCommand` Befehl geben.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
```

⚠ Important

Sie sollten Submodule nicht in Module importieren. Beispielsweise kann der folgende Code zu Fehlern führen.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/  
CognitoIdentity";
```

Der folgende Code ist korrekt.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Codegröße vergleichen

In Version 2 (v2) könnte ein einfaches Codebeispiel, das alle Ihre Amazon DynamoDB-Tabellen in der us-west-2 Region auflistet, wie folgt aussehen.

```
var AWS = require("aws-sdk");  
// Set the Region  
AWS.config.update({ region: "us-west-2" });  
// Create DynamoDB service object  
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });  
  
// Call DynamoDB to retrieve the list of tables  
ddb.listTables({ Limit: 10 }, function (err, data) {  
  if (err) {  
    console.log("Error", err.code);  
  } else {  
    console.log("Tables names are ", data.TableNames);  
  }  
});
```

v3 sieht wie folgt aus.

```
import {  
  DynamoDBClient,  
  ListTablesCommand  
} from "@aws-sdk/client-dynamodb";
```

```
const dbclient = new DynamoDBClient({ region: "us-west-2" });

try {
  const results = await dbclient.send(new ListTablesCommand);

  for (const item of results.TableNames) {
    console.log(item);
  }
} catch (err) {
  console.error(err)
}
```

Das `aws-sdk` Paket erweitert Ihre Anwendung um etwa 40 MB. Durch das `var AWS = require("aws-sdk")` Ersetzen durch `import {DynamoDB} from "@aws-sdk/client-dynamodb"` wird dieser Overhead auf etwa 3 MB reduziert. Durch die Beschränkung des Imports auf den DynamoDB-Client und den `ListTablesCommand` Befehl wird der Overhead auf weniger als 100 KB reduziert.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Befehle in Version 3 aufrufen

Sie können Operationen in Version 3 entweder mit v2- oder v3-Befehlen ausführen. Um v3-Befehle zu verwenden, importieren Sie die Befehle und die erforderlichen AWS Services-Paketclients und führen den Befehl mithilfe der `.send` Methode aus, die das `async/await`-Muster verwendet.

Um v2-Befehle zu verwenden, importieren Sie die erforderlichen AWS Services-Pakete und führen den v2-Befehl direkt im Paket aus, indem Sie entweder ein Callback- oder ein `Async/Await`-Muster verwenden.

Verwenden von v3-Befehlen

v3 stellt für jedes AWS Servicepaket eine Reihe von Befehlen bereit, mit denen Sie Operationen für diesen AWS Dienst ausführen können. Nachdem Sie einen AWS Dienst installiert haben, können Sie die verfügbaren Befehle in Ihrem Projekt durchsuchen `node_modules/@aws-sdk/client-PACKAGE_NAME/commands` folder.

Sie müssen die Befehle importieren, die Sie verwenden möchten. Der folgende Code lädt beispielsweise den DynamoDB-Dienst und den `CreateTableCommand` Befehl.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

Verwenden Sie die folgende Syntax, um diese Befehle im empfohlenen Async/Await-Muster aufzurufen.

```
CLIENT.send(new XXXCommand);
```

Im folgenden Beispiel wird beispielsweise eine DynamoDB-Tabelle mit dem empfohlenen `async/await`-Muster erstellt.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({ region: "us-west-2" });
const tableParams = {
  Table: TABLE_NAME
};

try {
  const data = await dynamodb.send(new CreateTableCommand(tableParams));
  console.log("Success", data);
} catch (err) {
  console.log("Error", err);
};
```

Verwenden von v2-Befehlen

Um v2-Befehle im SDK für zu verwenden JavaScript, importieren Sie die vollständigen AWS Servicepakete, wie im folgenden Code gezeigt.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Verwenden Sie die folgende Syntax, um v2-Befehle im empfohlenen Async/Await-Muster aufzurufen.

```
client.command(parameters);
```

Im folgenden Beispiel wird der `createTable` v2-Befehl verwendet, um eine DynamoDB-Tabelle mit dem empfohlenen `async/await`-Muster zu erstellen.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

Im folgenden Beispiel wird der `createBucket v2`-Befehl verwendet, um einen Amazon S3 S3-Bucket mithilfe des Callback-Musters zu erstellen.

```
const { S3 } = require('@aws-sdk/client-s3');
const s3 = new S3({ region: 'us-west-2' });
var bucketParams = {
  Bucket : BUCKET_NAME
};
function run() {
  s3.createBucket(bucketParams, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.Location);
    }
  })
};
run();
```

Neuer Middleware-Stack

Version 2 des SDK ermöglichte es Ihnen, eine Anfrage in den verschiedenen Phasen ihres Lebenszyklus zu ändern, indem Sie der Anfrage Event-Listener anhängen. Dieser Ansatz kann es schwierig machen, Fehler zu debuggen, die während des Lebenszyklus einer Anfrage schief gelaufen sind.

In Version 3 können Sie einen neuen Middleware-Stack verwenden, um den Lebenszyklus eines Operationsaufrufs zu steuern. Dieser Ansatz bietet eine Reihe von Vorteilen. Jede Middleware-Phase im Stack ruft die nächste Middleware-Stufe auf, nachdem Änderungen am Anforderungsobjekt vorgenommen wurden. Dies erleichtert auch das Debuggen von Problemen im Stack erheblich, da Sie genau sehen können, welche Middleware-Stufen aufgerufen wurden, bevor der Fehler aufgetreten ist.

Das folgende Beispiel fügt einem Amazon DynamoDB-Client (den wir zuvor erstellt und gezeigt haben) mithilfe von Middleware einen benutzerdefinierten Header hinzu. Das erste Argument ist eine Funktion, die akzeptiert `next`, was die nächste aufzurufende Middleware-Stufe im Stack ist, und ein `ObjectContext`, das einige Informationen über die aufgerufene Operation enthält. Die Funktion gibt eine Funktion zurück, die akzeptiert `args`, d. h. ein Objekt, das die an den Vorgang und die Anforderung übergebenen Parameter enthält. Sie gibt das Ergebnis des Aufrufs der nächsten Middleware mit `zurück. args` zurück.

```
dbclient.middlewareStack.add(  
  (next, context) => args => {  
    args.request.headers["Custom-Header"] = "value";  
    return next(args);  
  },  
  {  
    name: "my-middleware",  
    override: true,  
    step: "build"  
  }  
);  
  
dbclient.send(new PutObjectCommand(params));
```

Was ist der Unterschied zwischen AWS SDK for JavaScript v2 und v3

In diesem Abschnitt werden die wichtigsten Änderungen von AWS SDK for JavaScript Version 2 zu Version 3 beschrieben. Da v3 eine modulare Neufassung von v2 ist, unterscheiden sich einige grundlegende Konzepte zwischen v2 und v3. In unseren [Blogbeiträgen](#) erfahren Sie mehr über diese Änderungen. Die folgenden Blogbeiträge werden Sie auf den neuesten Stand bringen:

- [Modulare Pakete in AWS SDK for JavaScript](#)
- [Wir stellen vor: Middleware Stack in Modular AWS SDK for JavaScript](#)

Die Zusammenfassung der Schnittstellenänderungen von AWS SDK for JavaScript v2 zu v3 ist unten angegeben. Ziel ist es, Ihnen zu helfen, auf einfache Weise die v3-Äquivalente der v2-APIs zu finden, mit denen Sie bereits vertraut sind.

Themen

- [Konstrukteure des Kunden](#)
- [Anbieter von Anmeldeinformationen](#)
- [Überlegungen zu Amazon S3](#)
- [DynamoDB-Dokumentenclient](#)
- [Kellner und Unterzeichner](#)
- [Hinweise zu bestimmten Servicekunden](#)

Konstrukteure des Kunden

Diese Liste ist nach [v2-Konfigurationsparametern](#) indexiert.

- [computeChecksums](#)
 - v2: Ob MD5-Prüfsummen für Nutzlastkörper berechnet werden sollen, wenn der Dienst sie akzeptiert (derzeit nur in S3 unterstützt).
 - v3: Die entsprechenden Befehle von S3 (PutObject PutBucketCors, usw.) berechnen automatisch die MD5-Prüfsummen für die Nutzdaten der Anfrage. Sie können auch einen anderen Prüfsummenalgorithmus im ChecksumAlgorithm Parameter der Befehle angeben, um einen anderen Prüfsummenalgorithmus zu verwenden. [Weitere Informationen finden Sie in der Ankündigung der S3-Funktion](#)
- [convertResponseTypes](#)
 - v2: Ob Typen beim Parsen von Antwortdaten konvertiert werden.
 - v3: Veraltet. Diese Option gilt als nicht typsicher, da sie Typen wie Timestamp oder Base64-Binärdateien nicht aus der JSON-Antwort konvertiert.
- [correctClockSkew](#)
 - v2: Ob eine Korrektur der Zeitversetzung angewendet und Anfragen wiederholt werden sollen, die aufgrund einer schiefen Client-Uhr fehlschlagen.
 - v3: Veraltet. Das SDK wendet immer eine Korrektur der Taktverzerrung an.
- [systemClockOffset](#)
 - v2: Ein Offsetwert in Millisekunden, der für alle Signierzeiten gilt.
 - v3: Keine Änderung.

- [credentials](#)
 - v2: Die AWS Anmeldeinformationen, mit denen Anfragen signiert werden sollen.
 - v3: Keine Änderung. Es kann sich auch um eine asynchrone Funktion handeln, die Anmeldeinformationen zurückgibt. Wenn die Funktion ein `zurückgibtexpiration` (Date), wird die Funktion erneut aufgerufen, wenn sich das Ablaufdatum nähert. Anmeldeinformationen finden Sie in der [v3-Referenz. AwsAuthInputConfig](#)
- [endpointCacheSize](#)
 - v2: Die Größe des globalen Caches, in dem Endpunkte aus Endpoint Discovery-Vorgängen gespeichert werden.
 - v3: Keine Änderung.
- [endpointDiscoveryEnabled](#)
 - v2: Ob Operationen mit vom Dienst angegebenen Endpunkten dynamisch aufgerufen werden sollen.
 - v3: Keine Änderung.
- [hostPrefixEnabled](#)
 - v2: Ob Anforderungsparameter dem Präfix des Hostnamens zugewiesen werden sollen.
 - v3: Veraltet. Das SDK fügt bei Bedarf immer das Hostnamenpräfix ein.
- [httpOptions](#)

Eine Reihe von Optionen, die an die Low-Level-HTTP-Anfrage übergeben werden. Diese Optionen sind in Version 3 unterschiedlich aggregiert. Sie können sie konfigurieren, indem Sie eine neue `requestHandler` angeben. Hier ist das Beispiel für die Einstellung von HTTP-Optionen in der Laufzeit von Node.js. Weitere Informationen finden Sie in der [V3-Referenz für NodeHttpHandler](#).

Alle v3-Anfragen verwenden standardmäßig HTTPS. Sie müssen nur einen benutzerdefinierten `HTTPSAgent` bereitstellen.

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpsAgent: new Agent({
      /*params*/
    }),
    connectionTimeout: /*number in milliseconds*/,
    socketTimeout: /*number in milliseconds*/
  })
});
```

```
});
```

Wenn Sie einen benutzerdefinierten Endpunkt übergeben, der http verwendet, müssen Sie `HttpAgent` angeben.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: new Agent({
      /*params*/
    }),
  }),
  endpoint: "http://example.com",
});
```

Wenn der Client in Browsern ausgeführt wird, ist ein anderer Satz von Optionen verfügbar. Weitere Informationen finden Sie in der [v3-Referenz für FetchHttpHandler](#).

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new FetchHttpHandler({
    requestTimeout: /* number in milliseconds */
  }),
});
```

Jede Option von `httpOptions` ist unten angegeben:

- `proxy`
 - v2: Die URL, über die Anfragen weitergeleitet werden
 - v3: Sie können einen Proxy mit einem Agenten wie folgt [einrichten: Proxys für Node.js konfigurieren](#)
- `agent`
 - v2: Das Agent-Objekt, mit dem HTTP-Anfragen ausgeführt werden sollen. Wird für das Verbindungspooling verwendet.
 - v3: Sie können `httpAgent` oder `httpsAgent` wie in den obigen Beispielen gezeigt konfigurieren.
- `connectTimeout`

- v2: Setzt den Socket auf Timeout, wenn nach ConnectTimeout-Millisekunden keine Verbindung zum Server hergestellt werden konnte.
- [v3: ist in den Optionen verfügbar. `connectionTimeoutNodeHttpHandler`](#)
- `timeout`
 - v2: Die Anzahl der Millisekunden, die eine Anfrage dauern kann, bevor sie automatisch beendet wird.
 - v3: [socketTimeout ist in den Optionen verfügbar. `NodeHttpHandler`](#)
- `xhrAsync`
 - v2: Ob das SDK asynchrone HTTP-Anfragen sendet.
 - v3: Veraltet. Anfragen sind immer asynchron.
- `xhrWithCredentials`
 - v2: Legt die Eigenschaft „withCredentials“ eines HttpRequest XML-Objekts fest.
 - v3: Nicht verfügbar. Das SDK erbt [die Standard-Abrufkonfigurationen](#)
- [logger](#)
 - v2: Ein Objekt, das auf `.write ()` (wie ein Stream) oder `.log ()` (wie das Konsolenobjekt) reagiert, um Informationen über Anfragen zu protokollieren.
 - v3: Keine Änderung. Detailliertere Protokolle sind in Version 3 verfügbar.
- [maxRedirects](#)
 - v2: Die maximale Anzahl an Weiterleitungen, denen bei einer Serviceanfrage gefolgt werden muss.
 - v3: Veraltet. Das SDK folgt keinen Weiterleitungen, um unbeabsichtigte regionsübergreifende Anfragen zu vermeiden.
- [maxRetries](#)
 - v2: Die maximale Anzahl an Wiederholungen, die für eine Serviceanfrage ausgeführt werden können.
 - v3: Geändert zu. `maxAttempts` Weitere Informationen finden Sie in der [v3-Referenz für `RetryInputConfig`](#). Beachten Sie, dass das sein `maxAttempt` sollte `maxRetries + 1`.
- [paramValidation](#)
 - v2: Ob Eingabeparameter vor dem Senden der Anfrage anhand der Vorgangsbeschreibung überprüft werden sollen.
 - v3: Veraltet. Das SDK führt zur Laufzeit keine Validierung auf der Clientseite durch.
- [region](#)
 - v2: Die Region, an die Serviceanfragen gesendet werden sollen.
 - v3: Keine Änderung. Es kann sich auch um eine asynchrone Funktion handeln, die eine Regionszeichenfolge zurückgibt.

- [retryDelayOptions](#)
 - v2: Eine Reihe von Optionen zur Konfiguration der Wiederholungsverzögerung bei wiederholten Fehlern.
 - v3: Veraltet. Das SDK unterstützt eine flexiblere Wiederholungsstrategie mit der `retryStrategy` Client-Konstruktor-Option. Weitere Informationen finden Sie [in](#) der V3-Referenz
- [s3BucketEndpoint](#)
 - v2: Ob der angegebene Endpunkt einen einzelnen Bucket adressiert (falsch, wenn er den Root-API-Endpunkt adressiert).
 - v3: Geändert zu `bucketEndpoint`. Weitere Informationen finden Sie in der [v3-Referenz für BucketEndpoint](#). Beachten Sie, dass `true`, wenn Sie den Anforderungsendpunkt im Anforderungsparameter `bucket` aufgeben, der ursprüngliche Endpunkt überschrieben wird. In Version 2 hingegen überschreibt der Anforderungsendpunkt im Client-Konstruktor den Bucket Anforderungsparameter.
- [s3DisableBodySigning](#)
 - v2: Ob die S3-Bodysignatur deaktiviert werden soll, wenn die Signaturversion v4 verwendet wird.
 - v3: Umbenannt in `applyChecksum`
- [s3ForcePathStyle](#)
 - v2: Ob URLs im Pfadstil für S3-Objekte erzwungen werden sollen.
 - v3: Umbenannt in `forcePathStyle`
- [s3UseArnRegion](#)
 - v2: Ob die Anforderungsregion mit der Region überschrieben werden soll, die aus dem ARN der angeforderten Ressource abgeleitet wird.
 - v3: Umbenannt in `useArnRegion`
- [s3UseEast1RegionalEndpoint](#)
 - v2: Wenn die Region auf 'us-east-1' gesetzt ist, ob eine S3-Anfrage an globale Endpunkte oder an regionale Endpunkte 'us-east-1' gesendet werden soll.
 - v3: Veraltet. Der S3-Client verwendet immer den regionalen Endpunkt, wenn die Region auf eingestellt ist. `us-east-1` Sie können die Region so einstellen, dass Anfragen `aws-global` an den globalen S3-Endpunkt gesendet werden.
- [signatureCache](#)
 - v2: Ob die Signatur, mit der Anfragen signiert werden sollen (außer Kraft der API-Konfiguration), zwischengespeichert wird.
 - v3: Veraltet. Das SDK speichert die Hash-Signaturschlüssel immer zwischen.
- [signatureVersion](#)

- v2: Die Signaturversion, mit der Anfragen signiert werden sollen (überschreibt die API-Konfiguration).
- v3: Veraltet. Die im v2-SDK unterstützte Signatur V2 ist veraltet AWS. v3 unterstützt nur die Signatur v4.
- [sslEnabled](#)
 - v2: Ob SSL für Anfragen aktiviert ist.
 - v3: Umbenannt in `tls`.
- [stsRegionalEndpoints](#)
 - v2: Ob eine STS-Anfrage an globale oder regionale Endpunkte gesendet werden soll.
 - v3: Veraltet. Der STS-Client verwendet immer regionale Endpunkte, wenn er auf eine bestimmte Region eingestellt ist. Sie können die Region auf einstellen, um eine Anfrage `aws-global` an den globalen STS-Endpunkt zu senden.
- [useAccelerateEndpoint](#)
 - v2: Ob der Accelerate-Endpunkt mit dem S3-Dienst verwendet werden soll.
 - v3: Keine Änderung.

Anbieter von Anmeldeinformationen

In Version 2 JavaScript stellt das SDK für eine Liste von Anmeldeinformationsanbietern zur Auswahl sowie eine Anbieterkette für Anmeldeinformationen bereit, die standardmäßig auf Node.js verfügbar ist und versucht, die AWS Anmeldeinformationen von allen gängigen Anbietern zu laden. Das SDK für JavaScript Version 3 vereinfacht die Benutzeroberfläche des Anmeldeinformationsanbieters und macht es einfacher, benutzerdefinierte Anbieter für Anmeldeinformationen zu verwenden und zu schreiben. Zusätzlich zu einer neuen Anbieterkette für Anmeldeinformationen bietet das SDK für JavaScript Version 3 eine Liste von Anbietern von Anmeldeinformationen, die ein Äquivalent zu Version 2 anbieten sollen.

Hier sind alle Anbieter von Anmeldeinformationen in Version 2 und ihre Entsprechungen in Version 3.

Standardanbieter für Anmeldeinformationen

Der Standardanbieter für Anmeldeinformationen ist die Art und Weise, wie das SDK die AWS Anmeldeinformationen JavaScript auflöst, wenn Sie keinen explizit angeben.

- v2: [CredentialProviderChain](#) In Node.js werden Anmeldeinformationen aus Quellen in der folgenden Reihenfolge aufgelöst:
 - [Umgebungsvariable](#)

- [Datei mit gemeinsam genutzten Anmeldeinformationen](#)
- [Anmeldeinformationen für den ECS-Container](#)
- [Externer Prozess wird gestartet](#)
- [OIDC-Token aus der angegebenen Datei](#)
- [Metadaten der EC2-Instanz](#)

Wenn einer der oben genannten Anmeldeinformationsanbieter die AWS Anmeldeinformationen nicht auflösen kann, greift die Kette auf den nächsten Anbieter zurück, bis ein gültiger Berechtigungsnachweis aufgelöst ist, und die Kette gibt einen Fehler aus, wenn alle Anbieter ausfallen.

In Browser- und React Native-Laufzeiten ist die Anmeldeinformationskette leer und die Anmeldeinformationen müssen explizit festgelegt werden.

- v3: [defaultProvider](#). Die Quellen der Anmeldeinformationen und die Reihenfolge der Fallbacks ändern sich in Version 3 nicht. [Es unterstützt AWS IAM Identity Center auch Anmeldeinformationen.](#)

Temporäre Anmeldeinformationen

- v2: [ChainableTemporaryCredentials](#) steht für temporäre Anmeldeinformationen, die von abgerufen wurden `AWS.STS`. Ohne zusätzliche Parameter werden die Anmeldeinformationen aus dem `AWS.STS.getSessionToken()` Vorgang abgerufen. Wenn eine IAM-Rolle bereitgestellt wird, wird der `AWS.STS.assumeRole()` Vorgang stattdessen verwendet, um Anmeldeinformationen für die Rolle abzurufen. `AWS.ChainableTemporaryCredentials` unterscheidet sich von `AWS.TemporaryCredentials` der Art und Weise, wie `MasterCredentials` und Aktualisierungen behandelt werden. `AWS.ChainableTemporaryCredentials` aktualisiert abgelaufene Anmeldeinformationen mithilfe der vom Benutzer übergebenen `MasterCredentials`, um die Verkettung von STS-Anmeldeinformationen zu unterstützen. Reduziert jedoch die `MasterCredentials` während der Instanziierung `AWS.TemporaryCredentials` rekursiv, sodass Anmeldeinformationen, für die temporäre Anmeldeinformationen erforderlich sind, nicht aktualisiert werden können.

Das Original [TemporaryCredentials](#) wurde in Version 2 zugunsten von veraltet. `ChainableTemporaryCredentials`

- v3: [fromTemporaryCredentials](#) Sie können vom `@aws-sdk/credential-providers` Paket `fromTemporaryCredentials()` aus anrufen. Ein Beispiel:


```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
  credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Amazon Cognito Cognito-Identitätsanmeldedaten

Laden Sie Anmeldeinformationen aus dem Amazon Cognito Identity Service, der normalerweise in Browsern verwendet wird.

- v2: [CognitoIdentityCredentials](#) Stellt Anmeldeinformationen dar, die mithilfe des Amazon Cognito Identity Service von STS Web Identity Federation abgerufen wurden.
- v3: [Cognito Identity Credential Provider](#) Das [@aws/credential-providers](#) Paket stellt zwei Funktionen des Anmeldeinformationsanbieters bereit, von denen [fromCognitoIdentity](#) eine Identitäts-ID und Aufrufe [fromCognitoIdentityPool](#) entgegennimmt `cognitoIdentity: GetCredentialsForIdentity`, während die andere eine Identitätspool-ID verwendet, beim ersten Aufruf `cognitoIdentity: GetId` aufruft und dann aufruft. `fromCognitoIdentity` Nachfolgende Aufrufe des letzteren werden nicht erneut aufgerufen. `GetId`

Der Anbieter implementiert den im [Amazon Cognito Developer Guide](#) beschriebenen „Simplified Flow“. Der „Classic Flow“, bei dem zuerst angerufen `cognito: GetOpenIdToken` und dann angerufen `sts: AssumeRoleWithWebIdentity` wird, wird nicht unterstützt. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-
identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

Anmeldeinformationen für EC2-Metadaten (IMDS)

Stellt Anmeldeinformationen dar, die vom Metadaten-Service auf einer Amazon EC2 EC2-Instance empfangen wurden.

- v2: [EC2MetadataCredentials](#)
- v3: [fromInstanceMetadata](#): Erstellt einen Anmeldeinformationsanbieter, der Anmeldeinformationen aus dem Amazon EC2 Instance Metadata Service bezieht.

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); //
// CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

ECS-Anmeldeinformationen

Stellt Anmeldeinformationen dar, die von einer angegebenen URL empfangen wurden. Dieser Anbieter fordert temporäre Anmeldeinformationen von einer URI an, die in der `AWS_CONTAINER_CREDENTIALS_FULL_URI` Umgebungsvariablen `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` oder angegeben ist.

- v2: `ECSCredentials` oder [RemoteCredentials](#).
- v3: [fromContainerMetadata](#) erstellt einen Anmeldeinformationsanbieter, der Anmeldeinformationen vom Amazon ECS Container Metadata Service bezieht.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

Anmeldeinformationen für das Dateisystem

- v2: [FileSystemCredentials](#) steht für Anmeldeinformationen aus einer JSON-Datei auf der Festplatte.
- v3: Veraltet. Sie können die JSON-Datei explizit lesen und sie dem Client zur Verfügung stellen. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

Anbieter von SAML-Anmeldeinformationen

- v2: [SAMLCredentials](#) steht für Anmeldeinformationen, die von der STS-SAML-Unterstützung abgerufen wurden.
- v3: Nicht verfügbar. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

Anmeldeinformationen für die gemeinsame Anmeldeinformationsdatei

Lädt Anmeldeinformationen aus der Datei mit gemeinsam genutzten Anmeldeinformationen (standardmäßig `~/.aws/credentials` oder definiert durch die `AWS_SHARED_CREDENTIALS_FILE` Umgebungsvariable). Diese Datei wird von verschiedenen AWS SDKs und Tools unterstützt. Weitere Informationen finden Sie im [Dokument mit den gemeinsam genutzten Konfigurations- und Anmeldedaten](#).

- v2: [SharedIniFileCredentials](#)
- v3: [fromIni](#).

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilepath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
  }),
```

```
});
```

Anmeldeinformationen für die Web-Identität

Ruft Anmeldeinformationen mithilfe des OIDC-Tokens aus einer Datei auf der Festplatte ab. Es wird häufig in EKS verwendet.

- v2: [TokenFileWebIdentityCredentials](#).
- v3: [fromTokenFile](#)

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Anmeldeinformationen für Web Identity Federation

Ruft Anmeldeinformationen von der STS-Unterstützung für Web Identity Federation ab.

- v2: [WebIdentityCredentials](#)
- v3: [fromWebToken](#)

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
```

```
// Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
roleSessionName: "session:a",
// Optional. STS client config to make the assume role request.
clientConfig: { region },
}),
});
```

Überlegungen zu Amazon S3

Mehrteiliger Amazon S3 S3-Upload

In Version 2 enthält der Amazon S3-Client einen [upload\(\)](#)Vorgang, der das Hochladen großer Objekte mit der [von Amazon S3 angebotenen Funktion zum mehrteiligen Hochladen](#) unterstützt.

In Version 3 ist das [@aws-sdk/lib-storage](#)Paket verfügbar. Es unterstützt alle Funktionen, die im `upload()` v2-Betrieb angeboten werden, und unterstützt sowohl Node.js als auch die Browser-`Runtime`.

Vorsignierte Amazon S3 S3-URL

In Version 2 enthält der Amazon S3-Client die [getSignedUrlPromise\(\)](#)Operationen [getSignedUrl\(\)](#)und zum Generieren einer URL, mit der Benutzer Objekte von Amazon S3 hoch- oder herunterladen können.

In Version 3 ist das [@aws-sdk/s3-request-presigner](#)Paket verfügbar. Dieses Paket enthält die Funktionen für beide `getSignedUrl()` `getSignedUrlPromise()` Operationen. In diesem [Blogbeitrag](#) werden die Details dieses Pakets erörtert.

Amazon S3 S3-Regionsumleitungen

Wenn eine falsche Region an den Amazon S3-Client übergeben wird und ein darauffolgender Fehler `PermanentRedirect` (Status 301) ausgelöst wird, unterstützt der Amazon S3 S3-Client in Version 3 Regionsumleitungen (früher bekannt als Amazon S3 Global Client in Version 2). Sie können das [followRegionRedirects](#)Flag in der Client-Konfiguration verwenden, damit der Amazon S3 S3-Client regionalen Weiterleitungen folgt und seine Funktion als globaler Client unterstützt.

Note

Beachten Sie, dass diese Funktion zu zusätzlicher Latenz führen kann, da fehlgeschlagene Anfragen mit einer korrigierten Region erneut versucht werden, wenn ein

PermanentRedirect Fehler mit dem Status 301 empfangen wird. Diese Funktion sollte nur verwendet werden, wenn Sie die Region Ihrer Buckets nicht im Voraus kennen.

Amazon S3 S3-Streaming und gepufferte Antworten

Das v3-SDK zieht es vor, potenziell große Antworten nicht zu puffern. Dies tritt häufig bei der Amazon S3 GetObject S3-Operation auf, bei der Buffer in Version 2 a zurückgegeben wurde, Stream in Version 3 jedoch a zurückgegeben wurde.

Für Node.js müssen Sie den Stream verwenden oder den Client oder seinen Request-Handler löschen, um die Verbindungen für neuen Datenverkehr offen zu halten, indem Sie Sockets freigeben.

```
// v2
const get = await s3.getObject({ ... }).promise(); // this buffers consumes the stream
already.
```

```
// v3, consume the stream to free the socket
const get = await s3.getObject({ ... }); // object .Body has unconsumed stream
const str = await get.Body.transformToString(); // consumes the stream

// other ways to consume the stream include writing it to a file,
// passing it to another consumer like an upload, or buffering to
// a string or byte array.
```

Weitere Informationen finden Sie im Abschnitt zur [Socket-Erschöpfung](#).

DynamoDB-Dokumentenclient

Grundlegende Verwendung des DynamoDB-Dokumentenclients in Version 3

- In Version 2 können Sie die [AWS.DynamoDB.DocumentClient](#) Klasse verwenden, um DynamoDB-APIs mit systemeigenen JavaScript Typen wie Array, Number und Object aufzurufen. Es vereinfacht somit die Arbeit mit Elementen in Amazon DynamoDB, indem der Begriff der Attributwerte weggelassen wird.
- In Version 3 ist der entsprechende [@aws-sdk/lib-dynamodb](#) Client verfügbar. Es ähnelt normalen Service-Clients aus dem v3-SDK, mit dem Unterschied, dass es einen einfachen DynamoDB-Client in seinem Konstruktor verwendet.

Beispiel:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb"; // ES6 import
// const { DynamoDBClient } = require("@aws-sdk/client-dynamodb"); // CommonJS import
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb"; // ES6
import
// const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb"); //
CommonJS import

// Bare-bones DynamoDB Client
const client = new DynamoDBClient({});

// Bare-bones document client
const ddbDocClient = DynamoDBDocumentClient.from(client); // client is DynamoDB client

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "1",
      content: "content from DynamoDBDocumentClient",
    },
  })
);
```

UndefinedWerte in beim Marshalling

- In Version 2 wurden undefined Werte in Objekten beim Marshalling-Prozess für DynamoDB automatisch weggelassen.
- In Version 3 `@aws-sdk/lib-dynamodb` hat sich das Standard-Marshalling-Verhalten geändert: Objekte mit undefined Werten werden nicht mehr weggelassen. Um der Funktionalität von v2 gerecht zu werden, müssen Entwickler den `removeUndefinedValues` Wert `true` im `DynamoDB Document Client` explizit auf setzen. `marshallOptions`

Beispiel:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
```



```
// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted
    }
  })
);
```

Weitere Beispiele und Konfigurationen sind im [Paket README](#) verfügbar.

Kellner und Unterzeichner

Auf dieser Seite wird die Verwendung von Kellnern und Unterzeichnern in Version 3 beschrieben.

AWS SDK for JavaScript

Waiter

In Version 2 sind alle Kellner an die Klasse Service Client gebunden, und Sie müssen in der Eingabe des Kellners angeben, auf welchen festgelegten Status der Kunde warten wird. Beispielsweise müssen Sie anrufen, um darauf [waitFor\("bucketExists"\)](#) zu warten, dass ein neu erstellter Bucket bereit ist.

In Version 3 müssen Sie keine Kellner importieren, wenn Ihre Anwendung keinen benötigt. Darüber hinaus können Sie nur den Kellner importieren, den Sie benötigen, um auf den gewünschten Bundesstaat zu warten. So können Sie Ihre Paketgröße reduzieren und die Leistung verbessern. Hier ist ein Beispiel dafür, wie Sie darauf warten, dass der Bucket nach der Erstellung bereit ist:

```
import { S3Client, CreateBucketCommand, waitUntilBucketExists } from "@aws-sdk/client-s3"; // ES6 import
// const { S3Client, CreateBucketCommand, waitUntilBucketExists } = require("@aws-sdk/client-s3"); // CommonJS import

const Bucket = "BUCKET_NAME";
```

```
const client = new S3Client({ region: "REGION" });
const command = new CreateBucketCommand({ Bucket });

await client.send(command);
await waitUntilBucketExists({ client, maxWaitTime: 60 }, { Bucket });
```

Alles zur Konfiguration der Kellner findest du im [Blogbeitrag von waiters in the AWS SDK for JavaScript v3](#).

CloudFront Amazon-Unterzeichner

In Version 2 können Sie die Anfrage für den Zugriff auf eingeschränkte CloudFront Amazon-Distributionen mit [AWS.CloudFront.Signer](#) unterschreiben.

In Version 3 stehen Ihnen dieselben Dienstprogramme zur Verfügung, die im [@aws-sdk/cloudfront-signer](#) Paket enthalten sind.

Amazon RDS-Unterzeichner

In Version 2 können Sie das Authentifizierungstoken für eine Amazon RDS-Datenbank mithilfe von [AWS.RDS.Signer](#) generieren.

In Version 3 ist die ähnliche Dienstprogrammklasse im [@aws-sdk/rds-signer](#) Paket verfügbar.

Amazon Polly Signer

In Version 2 können Sie eine signierte URL für die vom Amazon Polly Polly-Service synthetisierte Sprache mit generieren. [AWS.Polly.Presigner](#)

In Version 3 ist eine ähnliche Hilfsfunktion im [@aws-sdk/polly-request-presigner](#) Paket verfügbar.

Hinweise zu bestimmten Servicekunden

AWS Lambda

Der Antworttyp für Lambda-Aufrufe unterscheidet sich in Version 2 und Version 3.

```
// v2
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";
```

```
const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

MD5-Prüfsumme

Um die Berechnung der MD5-Prüfsummen von Nachrichtentexten zu überspringen, setzen Sie den Wert für das `md5` Konfigurationsobjekt auf `false`. Andernfalls berechnet das SDK standardmäßig die Prüfsumme für das Senden von Nachrichten und validiert die Prüfsumme für abgerufene Nachrichten.

```
// Example: Skip MD5 checksum in Amazon SQS
import { SQS } from "@aws-sdk/client-sqs";
```

```
new SQS({
  md5: false // note: only available in v3.547.0 and higher
});
```

Bei der Verwendung eines benutzerdefinierten Elements `QueueUrl` in Amazon SQS SQS-Vorgängen, die diesen als Eingabeparameter verwenden, war es in Version 2 möglich, einen benutzerdefinierten Wert anzugeben, der den `QueueUrl` Standardendpunkt des Amazon SQS SQS-Clients überschreiben würde.

Nachrichten aus mehreren Regionen

In Version 3 sollten Sie einen Client pro Region verwenden. Die AWS Region soll auf Client-Ebene initialisiert und nicht zwischen Anfragen geändert werden.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqsClients = {
  "us-east-1": new SQS({ region: "us-east-1" }),
  "us-west-2": new SQS({ region: "us-west-2" }),
};

const queues = [
  { region: "us-east-1", url: "https://sqs.us-east-1.amazonaws.com/{AWS_ACCOUNT}/MyQueue" },
  { region: "us-west-2", url: "https://sqs.us-west-2.amazonaws.com/{AWS_ACCOUNT}/MyOtherQueue" },
];

for (const { region, url } of queues) {
  const params = {
    MessageBody: "Hello",
    QueueUrl: url,
  };
  await sqsClients[region].sendMessage(params);
}
```

Benutzerdefinierter Endpunkt

Wenn Sie in Version 3 einen benutzerdefinierten Endpunkt verwenden, d. h. einen, der sich von den standardmäßigen öffentlichen Amazon SQS SQS-Endpunkten unterscheidet, sollten Sie sowohl den Endpunkt auf dem Amazon SQS SQS-Client als auch das Feld festlegen. `QueueUrl`

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
  // for your region.
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
  // emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});
```

Wenn Sie keinen benutzerdefinierten Endpunkt verwenden, müssen Sie ihn nicht endpoint auf dem Client einrichten.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  region: "us-west-2",
});

await sqs.sendMessage({
  QueueUrl: "https://sqs.us-west-2.amazonaws.com/1234567/MyQueue",
  Message: "hello",
});
```

Dokumentenverlauf für AWS SDK for JavaScript Version 3

Dokumentverlauf

In der folgenden Tabelle werden die wichtigen Änderungen in der Version V3 AWS SDK for JavaScript ab dem 20. Oktober 2020 beschrieben. Für Benachrichtigungen über Aktualisierungen dieser Dokumentation können Sie einen [RSS-Feed](#) abonnieren.

Änderung	Beschreibung	Datum
Ankündigung	Das obere Banner mit einer end-of-support Erinnerung für Internet Explorer 11 wurde aktualisiert.	23. September 2022
Kleinere Updates	Kleinere Aktualisierungen zur besseren Übersicht und zur Behebung defekter Links. Dem Referenzhandbuch für AWS SDKs und Tools wurden Links zur Sensibilisierung hinzugefügt.	22. August 2022
Erzwingen Sie eine TLS-Mindestversion	Es wurden Informationen zu TLS 1.3 hinzugefügt.	31. März 2022
Das AWS Lambda Tutorial wurde aktualisiert	Es wurde ein Tutorial hinzugefügt, das zeigt, wie eine browserbasierte Anwendung zum Senden von Daten an eine Amazon DynamoDB-Tabelle erstellt wird.	20. Oktober 2020
Das Thema „Anmeldeinformationen in Node.js festlegen“ wurde aktualisiert	Aktualisiertes Thema zum Einstellen von Anmeldeinformationen	20. Oktober 2020

	formationen in Node.js für AWS SDK for JavaScript V3.	
Migrieren Sie zu Version 3	Es wurde ein Thema hinzugefügt, das beschreibt, wie man zu AWS SDK for JavaScript Version 3 migriert.	20. Oktober 2020
Erste Schritte	Die Themen für die ersten Schritte im Browser und die ersten Schritte mit Node.js für AWS SDK for JavaScript V3 wurden aktualisiert.	20. Oktober 2020
Browser-Builder	Informationen zu AWS Browser Builder wurden entfernt, da sie für AWS SDK for JavaScript Version 3 nicht erforderlich sind.	20. Oktober 2020
Servicebeispiele für Amazon Transcribe aktualisiert	Aktualisierte Amazon Transcribe-Servicebeispiele für AWS SDK for JavaScript V3.	20. Oktober 2020
Servicebeispiele für Amazon Simple Notification Service aktualisiert	Die Service-Beispiele von Amazon Simple Notification Service für AWS SDK for JavaScript Version 3 wurden aktualisiert.	20. Oktober 2020
Servicebeispiele für Amazon Simple Email Service aktualisiert	Die Service-Beispiele von Amazon Simple Email Service für AWS SDK for JavaScript Version 3 wurden aktualisiert.	20. Oktober 2020

Amazon Redshift Redshift-Servicebeispiele aktualisiert	Aktualisierte Amazon Redshift Redshift-Servicebeispiele für AWS SDK for JavaScript V3.	20. Oktober 2020
Amazon Lex Lex-Serveisbeispiele aktualisiert	Die Amazon Lex Lex-Serveisbeispiele für AWS SDK for JavaScript V3 wurden aktualisiert.	20. Oktober 2020
Amazon DynamoDB-Servicebeispiele aktualisiert	Aktualisierte Amazon DynamoDB-Servicebeispiele für AWS SDK for JavaScript V3.	20. Oktober 2020
AWS Elemental MediaConvert Service-Beispiele wurden aktualisiert	AWS Elemental MediaConvert Servicebeispiele für AWS SDK for JavaScript V3 aktualisiert.	20. Oktober 2020
AWS Lambda Servicebeispiele aktualisiert	AWS Lambda Servicebeispiele für AWS SDK for JavaScript V3 aktualisiert.	20. Oktober 2020
AWS SDK for JavaScript V3-Entwicklerhandbuch	Veröffentlichte Vorabversion des AWS SDK for JavaScript V3 Developer Guide.	19. Oktober 2020