aws

用户指南

Amazon CloudWatch

Amazon CloudWatch: 用户指南

Table of Contents

什么是 Amazon CloudWatch？ 1
访问 CloudWatch 1
相关 AWS 服务 1
CloudWatch 的工作原理 2
概念 2
命名空间 3
指标 3
尺寸 4
解决方案 6
统计数据 6
单位 6
时间段 7
聚合 7
百分位数 8
告警 9
账单和成本 9
资源 10
开始设置 11
注册 AWS 账户 11
创建具有管理访问权限的用户 11
登录 Amazon CloudWatch 控制台 12
设置 AWS CLI 13
开始使用 14
查看预构建的跨服务控制面板 19
使服务不出现在跨服务控制面板中 20
查看单项 AWS 服务的预构建控制面板 21
查看资源组的预构建控制面板 22
CloudWatch 账单和成本 24
使用 Cost Explorer 分析 CloudWatch 成本和使用数据 24
可视化和分析 CloudWatch 成本和使用数据 24
使用 AWS 成本和使用情况报告 和 Athena 分析 CloudWatch 成本和使用数据 28
使用 AWS 成本和使用情况报告 和 Athena 分析成本和使用数据 29
优化和降低成本的最佳实践 32
CloudWatch 指标 32

CloudWatch 告警 40
CloudWatch Logs 42
控制面板 47
创建控制面板 48
CloudWatch 跨账户可观测性控制面板 49
跨账户跨区域的控制面板 50
利用 AWS Management Console 创建和使用跨账户跨区域的控制面板 50
以编程方式创建跨账户跨区域的控制面板 51
使用控制面板变量创建灵活的控制面板 54
控制面板变量的类型 54
教程：将函数名称作为变量创建 Lambda 控制面板 55
教程：创建使用正则表达式模式在区域之间切换的控制面板 56
将变量复制到其他控制面板 57
在 CloudWatch 控制面板上创建和使用小部件 58
添加或删除图表 58
在 CloudWatch 控制面板上手动绘制指标图表 61
编辑图表 62
将 Explorer 小组件添加到 CloudWatch 控制面板 69
添加或删除折线图小组件 71
添加或删除数字小组件 72
添加或删除量规图小组件 73
将自定义小组件添加到 CloudWatch 控制面板 74
添加或删除文本小组件 85
添加或删除警报小组件 86
添加或删除表格小组件 87
链接和取消链接图表 90
共享控制面板 91
共享控制面板所需的权限 92
授予与您共享控制面板的人员的权限 93
与特定用户共享单个控制面板 94
公开共享单个控制面板 95
使用 SSO 共享账户中的所有 CloudWatch 控制面板 96
为 CloudWatch 控制面板共享设置 SSO 96
查看已共享的控制面板的数量 97
查看您的哪些控制面板已共享 98
停止共享一个或多个控制面板 98

查看共享控制面板权限并更改权限范围 99
允许与您共享的人员查看复合告警 100
允许与您共享的人员查看日志表小组件 101
允许与您共享的人员查看自定义小组件 103
使用实时数据 104
查看动画控制面板 105
将控制面板添加到收藏夹列表 105
更改时间段覆盖设置或刷新间隔 106
更改时间范围或时区格式 107
指标 110
基本监控和详细监控 110
使用 CloudWatch Metrics Insights 查询您的指标 112
构建查询 114
查询组件和语法 114
为 Metrics Insights 查询创建告警 123
将 Metrics Insights 查询与指标数学配合使用 126
使用自然语言生成与更新 CloudWatch Metrics Insights 查询 127
SQL 推理 130
示例查询 131
Metrics Insights 限制 139
Metrics Insights 术语表 140
Metrics Insights 问题排查 140
使用 Metrics Explorer 按标签和属性监控资源 141
适合 Metrics Explorer 的 CloudWatch 代理配置 143
使用指标流 143
设置指标流 145
可以流式传输的统计数据 155
指标流操作和维护 156
使用 CloudWatch 指标监控您的指标流 156
CloudWatch 和 Firehose 之间的信任关系 158
指标流输出格式 159
故障排除 187
查看可用的指标 188
搜索可用指标 191
绘制指标的图表 192
绘制指标图表 193

将两个图表合并为一个 198
使用动态标签 199
修改图表的时间范围或时区格式 201
放大图表 204
修改图表的 Y 轴 205
从图表上的指标创建告警 206
使用异常检测 208
异常检测的工作原理 210
指标数学异常检测 210
使用指标数学 211
向 CloudWatch 图表中添加数学表达式 211
指标数学语法和函数 212
使用 IF 表达式 240
指标数学异常检测 244
在图表中使用搜索表达式 244
搜索表达式语法 245
搜索表达式示例 250
使用搜索表达式创建图表 253
获取指标的统计数据 255
CloudWatch 统计数据定义 255
获取特定资源的统计数据 259
跨资源聚合统计数据 263
按 Auto Scaling 组聚合统计数据 265
按 AMI 聚合统计数据 267
发布 自定义指标 269
高精度指标 269
使用维度 270
发布单一数据点 271
发布统计数据集 272
发布零值 272
停止发布指标 273
告警 274
指标告警状态 275
评估告警 275
告警操作 276
Lambda 警报操作 277

配置告警处理缺失数据的方式 281
在数据缺失时如何评估告警状态 283
高精度告警 286
针对数学表达式的告警 286
基于百分位数的告警和小数据样本 286
CloudWatch 告警的常见功能 286
AWS 服务的警报建议 287
查找并创建建议警报 288
建议的警报 290
根据指标触发警报 370
根据静态阈值创建告警 370
根据指标数学表达式创建告警 372
基于 Metrics Insights 查询创建告警 375
基于连接的数据来源创建警报 375
根据异常检测创建告警 378
修改异常检测模型 381
删除异常检测模型 382
根据日志触发警报 382
根据日志组指标筛选条件创建告警 383
组合警报 384
创建复合告警 387
抑制复合警报操作 389
根据警报更改执行操作 397
将警报更改通知用户 398
告警事件和 EventBridge 405
管理警报 418
编辑或删除 CloudWatch 告警 418
隐藏 Auto Scaling 警报 420
警报使用场景和示例 420
创建账单警报 421
创建 CPU 使用率告警 424
创建负载均衡器延迟告警 426
创建存储吞吐量告警 428
针对 AWS 数据库中的性能详情计数器指标创建警报 430
创建告警以停止、终止、重启或恢复 EC2 实例 433
警报和标记 440

应用程序监控（APM） 441
Application Signals 441
Application Signals 所需权限 443
启用 Application Signals 450
监控应用程序的运行状况 520
收集的标准应用程序指标 561
服务级别目标（SLO） 565
SLO 概念 566
创建 SLO 568
查看 SLO 状态并对其进行分类 569
编辑现有 SLO 571
删除 SLO 571
综合监控（Canary） 572
所需角色和权限 574
创建金丝雀 588
组 685
在本地测试 Canary 686
排查失败金丝雀的问题 707
金丝雀脚本示例代码 715
金丝雀和 X-Ray 跟踪 721
在 VPC 上运行金丝雀 722
加密金丝雀构件 723
查看金丝雀统计数据和详细信息 725
金丝雀发布的 CloudWatch 指标 727
编辑或删除金丝雀脚本 729
启动、停止、删除或更新多个金丝雀脚本的运行时 731
使用 Amazon EventBridge 监控金丝雀事件 732
CloudWatch RUM 736
使用 CloudWatch RUM 的 IAM 策略 739
设置应用程序以使用 CloudWatch RUM 740
配置 CloudWatch RUM Web 客户端 748
区域化 750
使用页面组 750
指定自定义元数据 751
发送自定义事件 757
查看 CloudWatch RUM 控制面板 759

您可以使用 CloudWatch RUM 收集的 CloudWatch 指标 761
CloudWatch RUM 的数据保护和数据隐私 771
CloudWatch RUM Web 客户端收集的信息 772
管理使用 CloudWatch RUM 的应用程序 803
CloudWatch RUM 配额 804
故障排除 805
使用 CloudWatch Evidently 执行启动和 A/B 实验 805
使用 Evidently 的 IAM policy 806
创建项目、功能、启动和实验 808
管理功能、启动和实验 826
将代码添加到应用程序 831
项目数据存储 833
Evidently 是如何计算结果的 836
在控制面板中查看启动结果 838
在控制面板中查看实验结果 838
CloudWatch Evidently 如何收集和存储数据 839
使用服务相关角色 840
CloudWatch Evidently 配额 842
教程：使用 Evidently 示例应用程序进行 A/B 测试 843
网络监控 854
使用 Internet Monitor 854
支持的区域 856
定价 857
组件 858
互联网天气地图 860
Internet Monitor 的工作原理 861
用例 867
网络监测仪跨账户可观测性 868
开始使用 868
使用 CLI 的示例 883
Internet Monitor 控制面板 892
使用工具探索数据 900
创建警报 919
EventBridge 集成 920
排查错误 920
数据保护和数据隐私 921

Identity and Access Management 922
配额 940
使用网络监测仪 941
网络监测仪主要功能 941
术语和组件 941
限制和要求 942
网络监测仪的工作原理 942
区域可用性 944
创建监测仪 945
使用监测仪和探测器 949
网络监测仪控制面板 956
配额 961
安全性 962
Identity and Access management 964
定价 980
基础设施监控 981
Container Insights 981
针对 Amazon EKS 增强了可观测性的 Container Insights 982
支持的平台 983
CloudWatch 代理容器镜像 983
设置 Container Insights 983
查看 Container Insights 指标 1039
Container Insights 收集的指标 1042
性能日志参考 1123
Container Insights Prometheus 指标监控 1156
与 Application Insights 集成 1279
在 Container Insights 中查看 Amazon ECS 生命周期事件 1279
Container Insights 问题排查 1280
构建您自己的 CloudWatch 代理 Docker 镜像 1284
在容器中部署其他 CloudWatch 代理功能 1284
Lambda Insights 1285
开始使用 Lambda Insights 1285
查看 Lambda Insights 指标 1349
与 Application Insights 集成 1350
Lambda Insights 收集的指标 1350
问题排查和已知问题 1353

示例遥测事件 1354
使用 Contributor Insights 分析高基数数据 1356
创建 Contributor Insights 规则 1357
Contributor Insights 规则语法 1362
示例规则 1366
查看 Contributor Insights 报告 1370
绘制规则生成的指标的图表 1371
使用 Contributor Insights 内置规则 1374
使用 CloudWatch Application Insights 检测常见应用程序问题 1374
Amazon CloudWatch Application Insights 是什么？ 1375
Application Insights 的工作方式 1383
开始使用 1398
Application Insights 跨账户可观测性 1428
使用组件配置 1429
使用 CloudFormation 模板 1499
教程：为 SAP ASE 设置监控 1512
教程：为 SAP HANA 设置监控 1520
教程：为 SAP NetWeaver 设置监控 1535
查看和排查 Application Insights 1551
支持的日志和指标 1555
使用资源运行状况视图 1648
先决条件 1648
跨账户和跨区域监控 1651
CloudWatch 跨账户可观测性 1653
将监控账户与源账户相关联 1655
管理监控账户和源账户 1665
跨账户跨区域的 CloudWatch 控制台 1668
启用跨账户跨区域功能 1669
（可选）与 AWS Organizations 集成 1672
故障排除 1673
使用跨账户后禁用和清理 1674
查询源自其他数据来源的指标 1675
管理对数据来源的访问 1676
通过向导连接到预构建数据来源 1676
Amazon Managed Service for Prometheus 1678
Amazon OpenSearch Service 1678

Amazon RDS for PostgreSQL 和 Amazon RDS for MySQL 1679
Amazon S3 CSV 文件 1680
Microsoft Azure Monitor 1681
Prometheus 1682
可用更新的通知 1683
创建自定义数据来源连接器。 1683
使用模板 1684
从头开始创建自定义数据来源 1685
使用您的自定义数据来源 1690
如何将参数传递给您的 Lambda 函数 1690
删除数据来源连接器 1691
使用 CloudWatch 代理收集指标、日志和跟踪信息 1692
安装 CloudWatch 代理 1694
使用命令行安装 CloudWatch 代理 1695
使用 AWS Systems Manager 安装 CloudWatch 代理 1716
在本地服务器上安装 CloudWatch 代理 1727
使用 AWS CloudFormation 在新实例上安装 CloudWatch 代理 1732
教程：使用 AWS CloudFormation 内联模板进行安装 1733
教程：使用 AWS CloudFormation 和 Parameter Store 安装 CloudWatch 1735
对利用 AWS CloudFormation 的 CloudWatch 代理安装进行故障排除 1737
CloudWatch 代理凭证首选项 1738
验证 CloudWatch 代理软件包的签名 1739
创建 CloudWatch 代理配置文件 1748
使用向导创建 CloudWatch 代理配置文件 1748
手动创建或编辑 CloudWatch 代理配置文件 1755
使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理 1840
选项 1：使用 IAM 权限在 Worker 节点上进行安装 1841
选项 2：使用 IAM 服务账户角色进行安装（仅适用于使用加载项的情况） 1843
（可选）其他配置 1844
故障排除 1853
CloudWatch 代理收集的指标 1854
Windows Server 实例上的 CloudWatch 代理收集的指标 1854
Linux 和 macOS 实例上的 CloudWatch 代理收集的指标 1855
内存指标定义 1867
CloudWatch 代理的常见场景 1870
以不同用户身份运行 CloudWatch 代理 1871

CloudWatch 代理如何处理稀疏日志文件 1872
将自定义维度添加到 CloudWatch 代理收集的指标 1873
多个 CloudWatch 代理配置文件 1873
汇总或累积 CloudWatch 代理收集的指标 1876
使用 CloudWatch 代理收集高精度指标 1877
向不同账户发送指标、日志和跟踪信息 1878
统一的 CloudWatch 代理和更早的 CloudWatch Logs 代理之间的时间戳差异 1880
CloudWatch 代理故障排除 1880
CloudWatch 代理命令行参数 1881
使用 Run Command 安装 CloudWatch 代理失败 1881
CloudWatch 代理无法启动 1881
验证 CloudWatch 代理是否正在运行 1882
CloudWatch 代理未启动，并且错误中提及了 Amazon EC2 区域 1883
无法在 Windows Server 上启动 CloudWatch 代理 1883
指标存储在何处？ 1884
CloudWatch 代理需要很长时间才能在容器中运行或记录跃点数限制错误 1884
我更新了代理配置，但在 CloudWatch 控制台中看不到新的指标或日志 1885
CloudWatch 代理文件和位置 1885
查找有关 CloudWatch 代理版本的信息 1887
CloudWatch 代理生成的日志 1888
停止和重新启动 CloudWatch 代理 1888
在日志中嵌入指标 1890
发布使用嵌入式指标格式的日志 1890
使用客户端库 1891
规范：嵌入式指标格式 1892
使用 PutLogEvents API 发送手动创建的嵌入式指标格式日志 1900
使用 CloudWatch 代理发送嵌入式指标格式日志 1902
通过 AWS Distro for OpenTelemetry 使用嵌入式指标格式 1908
在控制台中查看您的指标和日志 1909
为使用嵌入式指标格式创建的指标设置警报 1910
发布 CloudWatch 指标的服务 1911
AWS 使用情况指标 1927
可视化 Service Quotas 并设置告警 1927
AWS API 使用情况指标 1929
CloudWatch 使用情况指标 1937
CloudWatch 教程 1939

方案：监控估计费用 1939
步骤 1：启用账单提醒 1940
步骤 2：创建账单告警 1940
步骤 3：检查告警状态 1942
步骤 4：编辑账单告警 1942
步骤 5：删除账单告警 1943
方案：发布指标 1943
步骤 1：定义数据配置 1944
步骤 2：将指标添加到 CloudWatch 1944
步骤 3：获取 CloudWatch 中的统计数据 1945
步骤 4：使用控制台查看图表 1946
使用 AWS SDK 1947
代码示例 1949
基础知识 1955
开始使用 CloudWatch 1955
了解基础知识 1960
操作 2036
场景 2175
开始使用告警 2175
管理指标和告警 2178
监控 DynamoDB 性能 2186
安全性 2188
数据保护 2188
传输中加密 2189
Identity and Access Management 2189
受众 2190
使用身份进行身份验证 2190
使用策略管理访问 2193
Amazon CloudWatch 如何与 IAM 协同工作 2195
基于身份的策略示例 2200
故障排除 2205
CloudWatch 控制面板权限更新 2206
用于 CloudWatch 的 AWS 托管式（预定义）策略 2207
客户管理型策略示例 2238
策略更新 2240
使用条件键限制对 CloudWatch 命名空间的访问 2259

使用条件键限制 Contributor Insights 用户对日志组的访问 2260
使用条件键限制告警操作 2262
使用服务相关角色 2263
对 CloudWatch RUM 使用服务相关角色 2273
在 Application Insights 中使用服务相关角色 2278
适用于 Application Insights 的 AWS 托管式策略 2290
Amazon CloudWatch 权限参考 2300
合规性验证 2313
故障恢复能力 2314
基础设施安全性 2314
网络隔离 2314
AWS Security Hub 2315
接口 VPC 端点 2315
CloudWatch 2316
CloudWatch Synthetics 2317
Synthetics 金丝雀的安全注意事项 2319
使用安全连接 2319
金丝雀命名注意事项 2320
金丝雀代码中的密钥和敏感信息 2320
权限注意事项 2320
堆栈跟踪和异常消息 2321
缩小 IAM 角色的范围 2321
敏感数据编辑 2321
使用 AWS CloudTrail 记录 API 和控制台操作 2323
CloudTrail 中的 CloudWatch 信息 2324
示例：CloudWatch 日志文件条目 2325
CloudTrail 中的 CloudWatch 数据面板事件 2327
CloudTrail 中的查询生成信息 2330
CloudTrail 中的 CloudWatch Internet Monitor 2331
示例：CloudWatch Internet Monitor 日志文件条目 2332
CloudTrail 中的 CloudWatch Synthetics 信息 2334
示例：CloudWatch Synthetics 日志文件条目 2335
CloudTrail 中的 CloudWatch RUM 信息 2338
示例：CloudWatch RUM 日志文件条目 2339
CloudTrail 中的 CloudWatch RUM 数据面板事件 2343
标记 CloudWatch 资源 2346

CloudWatch 中支持的资源 2346
管理标签 2346
标签命名和使用约定 2347
Grafana 集成 2348
服务限额 2349
文档历史记录 2356

什么是 Amazon CloudWatch？

Amazon CloudWatch 可实时监控您的亚马逊云科技 (AWS) 资源以及您在 AWS 上运行的应用程序。您可以使用 CloudWatch 收集和跟踪指标，这些指标是您可衡量的相关资源和应用程序的变量。

CloudWatch 主页自动显示有关您使用的每项 AWS 服务的指标。此外，您还可以创建自定义控制面板，以显示有关自定义应用程序的指标，并显示您选择的指标的自定义集合。

您可以创建警报，这些警报监视指标，当超出阈值时，它们会发送通知或者对您所监控的资源自动进行更改。例如，您可以监控您的 Amazon EC2 实例的 CPU 使用率以及磁盘读写情况，然后使用此数据确定您是否应启动其它实例来处理增加的负载。您还可以使用此数据停止未完全利用的实例以节省开支。

您可通过使用 CloudWatch 了解系统范围的资源使用率、应用程序性能和运行状况。

访问 CloudWatch

您可以使用以下任何方式访问 CloudWatch：

- Amazon CloudWatch 控制台 – https://console.aws.amazon.com/cloudwatch/
- AWS CLI – 有关更多信息，请参阅 AWS Command Line Interface 用户指南中的开始设置 AWS Command Line Interface。
- CloudWatch API – 有关更多信息，请参阅 Amazon CloudWatch API 参考。
- AWS SDK – 有关更多信息，请参阅亚马逊云科技的工具。

相关 AWS 服务

以下服务可与 Amazon CloudWatch 一起使用：

- Amazon Simple Notification Service (Amazon SNS) 可协调和管理向订阅端点或客户端传送或发送消息的过程。您可结合使用 Amazon SNS 与 CloudWatch 以便在达到告警阈值时发送消息。有关更多信息，请参阅 设置 Amazon SNS 通知。
- Amazon EC2 Auto Scaling 支持根据用户定义的策略、运行状况检查和时间表自动启动或终止 Amazon EC2 实例。您可将 CloudWatch 警报与 Amazon EC2 Auto Scaling 一起使用以按需扩展 EC2 实例。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南中的动态扩缩。

- AWS CloudTrail 支持监控针对您的账户向 Amazon CloudWatch API 做出的调用（包括由 AWS Management Console、AWS CLI 和其他服务进行的调用）。开启 CloudTrail 日志记录后，CloudWatch 将日志文件写入到您在配置 CloudTrail 时指定的 Amazon S3 存储桶。有关更多信息，请参阅 使用 AWS CloudTrail 记录 Amazon CloudWatch API 和控制台操作。
- AWS Identity and Access Management (IAM) 是一种 Web 服务，可帮助您安全地控制用户对 AWS 资源的访问权限。使用 IAM 可以控制可使用您的 AWS 资源（身份验证）的人员、他们可以使用的资源以及使用这些资源的方式（授权）。有关更多信息，请参阅 适用于 Amazon CloudWatch 的 Identity and Access Management。

说明 Amazon CloudWatch 的工作原理

Amazon CloudWatch 本质上是一个指标存储库。AWS 服务（例如 Amazon EC2）会将指标放在存储库中，而您可以根据这些指标来检索统计数据。如果将自己的自定义指标放在存储库中，则还可以检索有关这些指标的统计数据。

您可以在 CloudWatch 控制台中使用指标计算统计数据，然后以图形化的方式显示数据。有关生成指标并将其发送到 CloudWatch 的其他 AWS 资源的更多信息，请参阅 发布 CloudWatch 指标的 AWS 服务。

在满足特定条件时，您可以配置告警操作以停止、启动或终止 Amazon EC2 实例。此外，您可创建代您启动 Amazon EC2 Auto Scaling 和 Amazon Simple Notification Service (Amazon SNS) 操作的告警。有关创建 CloudWatch 警报的更多信息，请参阅告警。

AWS 云计算资源存储在具有高度可用性的数据中心设施中。要提供额外的可扩展性和可靠性，每个数据中心设施应位于称为区域 的特定地理区域。从设计而言，每个区域都与其他区域完全隔离，以实现最大程度的故障隔离和稳定性。指标单独存储在区域中，但您可以使用 CloudWatch 跨区域功能来聚合来自不同区域的统计数据。有关更多信息，请参阅 Amazon Web Services 一般参考 中的 跨账户跨区域的 CloudWatch 控制台 以及区域和端点。

Amazon CloudWatch 的概念

下面是便于您了解和使用 Amazon CloudWatch 的核心术语和概念：

- 命名空间
- 指标
- 尺寸

- 解决方案
- 统计数据
- 百分位数
- 告警

有关 CloudWatch 指标、警报、API 请求和警报电子邮件通知的服务限额的信息，请参阅 CloudWatch 服务限额。

命名空间

命名空间是用于 CloudWatch 指标的容器。不同命名空间中的指标彼此独立，因此来自不同应用程序的指标不会被错误地聚合到相同的统计信息中。

无默认命名空间。您必须为发布到 CloudWatch 的每个数据点指定命名空间。在创建指标时，您可以指定命名空间名称。这些名称必须包含有效的 ASCII 字符，且长度必须等于或少于 255 个字符。可用字符包括：字母数字字符（0-9A-Za-z）、句点（.）、连字符（-）、下划线（_）、正斜杠（/）、井号（#）、冒号（:）和空格符号。一个命名空间必须包含至少一个非空格字符。

AWS 命名空间通常使用以下命名约定：`AWS/service`。例如，Amazon EC2 使用 `AWS/EC2` 命名空间。有关 AWS 命名空间的列表，请参阅 发布 CloudWatch 指标的 AWS 服务。

指标

指标是 CloudWatch 中的基本概念。指标表示一个发布到 CloudWatch 并按时间排序的数据点集。可将指标视为要监控的变量，而数据点代表该变量随时间变化的值。例如，特定 EC2 实例的 CPU 使用率是 Amazon EC2 提供的一个指标。数据点本身可来自于您从中收集数据的任何应用程序或业务活动。

预设情况下，许多 AWS 服务免费提供资源（例如 Amazon EC2 实例、Amazon EBS 卷和 Amazon RDS 数据库实例）的指标。如需收费，您也可以对某些资源（例如 Amazon EC2 实例）或您自己的应用程序指标启用详细监控。关于自定义指标，您可以您选择的任何顺序和任何速率添加数据点。您可以按一组有序的时间序列数据来检索关于这些数据点的统计数据。

指标仅存在于创建它们的区域中。指标无法删除，但如果在 15 个月后没有向指标发布新数据，这些指标将自动过期。依据滚动机制，15 个月之前的数据点将过期；当新的数据点进入时，15 个月之前的数据将被丢弃。

指标是通过一个名称、一个命名空间以及零个或多个维度进行唯一定义的。指标中的每个数据点都有一个时间戳和一个度量单位（可选）。您可以从 CloudWatch 中检索任何指标的统计数据。

有关更多信息，请参阅 查看可用的指标 和 发布 自定义指标。

时间戳

每个指标数据点必须与一个时间戳关联。时间戳最长可以为过去的两周和将来的两小时。如果不提供时间戳，CloudWatch 会根据收到数据点的时间创建一个时间戳。

时间戳为 `dateTime` 对象，包含完整的日期以及小时、分钟和秒 (例如，2016-10-31T23:59:59Z)。有关更多信息，请参阅 dateTime。尽管没有强制要求，但我们建议您使用协调世界时 (UTC)。从 CloudWatch 检索统计数据时，所有时间均采用 UTC。

CloudWatch 告警会基于当前时间 (UTC) 检查指标。发送到 CloudWatch 并且时间戳为非 UTC 时间的自定义指标会导致告警显示数据不足状态或产生延迟告警。

指标保留

CloudWatch 将保留指标数据，如下所示：

- 时间段短于 60 秒的数据点的可用时间为 3 小时。这些数据点是高精度自定义指标。
- 时间段为 60 秒 (1 分钟) 的数据点可用 15 天
- 时间段为 300 秒（5 分钟）的数据点可用 63 天
- 时间段为 3600 秒 (1 小时) 的数据点可用 455 天 (15 个月)

最初以较短时间段发布的数据点汇总在一起，可实现长期存储。例如，如果您使用 1 分钟的时间段收集数据，数据以 1 分钟的精度保持 15 天可用。15 天之后，此数据仍可用，但汇总在一起，只能以 5 分钟的精度检索。63 天之后，数据进一步汇总，以 1 小时的精度提供。

> **Note**
>
> 控制台中不会显示在过去两周内没有任何新数据点的指标。当您在控制台的 All metrics（全部指标）选项卡的搜索框中键入指标名称或维度名称时，它们也不会显示，并且 list-metrics 命令的结果中不会返回它们。检索这些指标的最佳方法是使用 AWS CLI 中的 get-metric-data 或者 get-metric-statistics 命令。

尺寸

维度是一个名称/值对，它是指标标识的一部分。您可以为一个指标分配最多 30 个维度。

每个指标包含用于描述该指标的特定特征，您可以将维度理解为这些特征的类别。维度可以帮助您设计统计数据计划的结构。因为维度是指标的唯一标识符的一部分，因此无论您在何时向一个指标添加唯一名称/值对，都会创建该指标的一个新变体。

向 CloudWatch 发送数据的 AWS 服务会向每个指标附加维度。您可使用维度筛选 CloudWatch 返回的结果。例如，您可通过在搜索指标时指定 `InstanceId` 维度来获取特定 EC2 实例的统计数据。

对于由特定 AWS 服务（如 Amazon EC2）生成的指标，CloudWatch 可以聚合多个维度的数据。例如，如果您在 `AWS/EC2` 命名空间中搜索指标但不指定任何维度，则 CloudWatch 会聚合指定指标的所有数据以创建您请求的统计数据。CloudWatch 不会跨多个维度聚合自定义指标的数据。

维度组合

CloudWatch 将维度的每种唯一组合视为一个单独的指标，即使指标具有相同的指标名称也是如此。您只能使用已发布的特定维度组合检索统计数据。当您检索统计数据时，为命名空间、指标名称和维度参数指定创建指标时使用的相同值。您还可指定 CloudWatch 要用于聚合的开始和结束时间。

例如，假设您在具有以下属性的 DataCenterMetric 命名空间中发布了 4 个名为 ServerStats 的不同指标：

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:30:00Z, Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:31:00Z, Value: 115
Dimensions: Server=Prod, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:32:00Z, Value: 95
Dimensions: Server=Beta, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:33:00Z, Value: 97
```

如果您仅发布这 4 个指标，则可检索这些维度组合的统计数据：

- `Server=Prod,Domain=Frankfurt`
- `Server=Prod,Domain=Rio`
- `Server=Beta,Domain=Frankfurt`
- `Server=Beta,Domain=Rio`

如果您未指定任何维度，则无法检索以下维度的统计数据。（使用指标数学 SEARCH 函数时例外，该函数可以检索多个指标的统计数据。有关更多信息，请参阅 在图表中使用搜索表达式。）

- `Server=Prod`

- `Server=Beta`
- `Domain=Frankfurt`
- `Domain=Rio`

解决方案

每个指标均为以下类型之一：

- 标准精度，数据粒度为一分钟
- 高精度，数据粒度为一秒

AWS 服务生成的指标在默认情况下为标准精度。在发布自定义指标时，您可以将其定义为标准精度或高精度。发布高精度指标时，CloudWatch 使用 1 秒的精度来存储指标，您可以按照 1 秒、5 秒、10 秒、30 秒或 60 秒的任意倍数的时间段读取和检索。

高精度指标让您对应用程序的亚分钟级活动有着更详细的直观认识。请记住，每次对自定义指标的 `PutMetricData` 调用都会收取费用，因此对高精度指标频繁调用 `PutMetricData` 会导致较高的费用。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

如果对高精度指标设置告警，您可以指定 10 秒或 30 秒时间段的高精度告警，也可以设置 60 秒的任意倍数时间段的定期告警。10 秒或 30 秒时间段的高精度警报会产生较高的费用。

统计数据

统计数据是指定时间段内的指标数据汇总。CloudWatch 提供统计数据的依据是您的自定义数据所提供的指标数据点，或其他 AWS 服务向 CloudWatch 提供的指标数据点。聚合通过使用命名空间、指标名称、维度以及数据点度量单位在您指定的时间段内完成。

有关 CloudWatch 支持的统计数据的详细定义，请参阅 CloudWatch 统计数据定义。

单位

所有统计数据都有度量单位。示例单位包括 `Bytes`、`Seconds`、`Count` 和 `Percent`。有关 CloudWatch 支持的单位的完整列表，请参阅 Amazon CloudWatch API 参考中的 MetricDatum 数据类型。

您可在创建自定义指标时指定单位。如果未指定单位，CloudWatch 将使用 None 作为单位。单位帮助为数据提供概念意义。尽管在内部单位对于 CloudWatch 无重大意义，但其他应用程序还是可以根据单位得到语义信息。

指定度量单位的指标数据点是单独聚合的。当获得统计数据而不指定单位时，CloudWatch 会将相同单位的所有数据点聚合在一起。如果有两个完全相同的指标包含不同的单位，则每个单位会各返回一个数据流。

时间段

时间段是与特定 Amazon CloudWatch 统计数据关联的时间的长度。每项统计信息代表在指定时间段内对收集的指标数据的聚合。时间段以秒为单位定义，时间段的有效值为 1、5、10、30 或 60 的任意倍数。例如，要指定六分钟的时间段，时间段的值应为 360。通过改变时间段的长度可以调整数据聚合的方式。该期限的默认值为 60 秒。期限可以短至一秒，如果期限大于默认值 60 秒，则必须为 60 的倍数。

只有您使用 1 秒的存储精度定义的自定义指标支持亚分钟级时间段。虽然设置为低于 60 的时间段的选项在控制台中始终可用，但您应选择与指标存储方式相符的时间段。有关支持亚分钟级时间段的指标的更多信息，请参阅高精度指标。

当您检索统计数据时，可指定时间段、开始时间和结束时间。这些参数决定了与统计数据关联的时间的总长度。开始时间和结束时间的默认值将让您获得上一小时的统计数据。您指定的开始时间值和结束时间值将确定 CloudWatch 返回的时间段的数量。例如，使用时间段、开始时间和结束时间的默认值检索统计数据将返回上一小时内每分钟的聚合统计数据集。如果您想要以 10 分钟为一块来聚合统计信息，请指定时间段 600。对于一个完整小时内聚合的统计数据，请指定时间段 3600。

当对某个时段的统计数据进行聚合时，将为聚合的数据标记上对应的时段起始时间。例如，从晚上 7:00 到晚上 8:00 聚合的数据将会加上晚上 7:00 的时间戳。此外，在晚上 7:00 和晚上 8:00 之间聚合的数据将在晚上 7:00 开始可见，然后该聚合数据的值可能会随着 CloudWatch 收集该时间段内的更多样本而发生更改。

时间段对 CloudWatch 告警也很重要。当您创建告警以监控特定指标时，您是在要求 CloudWatch 将该指标与指定的阈值进行比较。您对 CloudWatch 如何进行此比较拥有广泛的控制力。您不仅能够指定进行比较的时间段，还能够指定用于得出结论的评估期间的数目。例如，如果您指定三个评估期间，则 CloudWatch 将比较三个数据点的时间段。CloudWatch 仅告知您最旧的数据点是否超出阈值以及其他数据点是否超出阈值或丢失。

聚合

Amazon CloudWatch 将根据您在检索统计数据时指定的时间段长度聚合统计数据。您可以根据需要发布任意数量包含相同或类似时间戳的数据点。CloudWatch 会根据指定周期长度聚合这些数据点。CloudWatch 不会跨区域自动聚合数据，但您可以使用指标数学来聚合来自不同区域的指标。

您可以为具有相同时间戳、相同命名空间和维度的指标发布数据点。CloudWatch 会返回这些数据点的聚合统计数据。还可以为包含任意时间戳的相同或不同指标发布多个数据点。

对于大型数据集，您可插入称为统计数据集的预先汇总数据集。通过统计数据集，可以让 CloudWatch 为一定数量的数据点提供 Min、Max、Sum 和 SampleCount。当您需要在一分钟内多次收集数据时，通常可以使用它。例如，假设您拥有一个关于网页请求延迟的指标。对命中的每一个网页都发布数据毫无意义。建议您收集该网页的所有点击延迟，每分钟汇总一次，然后将统计数据集发送到 CloudWatch。

Amazon CloudWatch 不会区分指标的来源。如果从不同的来源发布包含相同命名空间和维度的一个指标，则 CloudWatch 会将其视为一个单独的指标。这对分布式扩展型系统中的服务指标有其作用。例如，Web 服务器应用程序中的所有主机都可以发布表示其正在处理的请求延迟的相同指标。CloudWatch 会将这些指标视为单一指标，以便您可以获取应用程序中所有请求的最小值、最大值、平均值和总和的统计数据。

百分位数

百分位数指示某个值在数据集中的相对位置。例如，第 95 个百分位数表示 95% 的数据低于此值，5% 的数据高于此值。百分位数可帮助您更好地了解指标数据的分布情况。

百分位数通常用于隔离异常值。在正态分布中，95% 的数据在平均值的两个标准偏差范围内，99.7% 的数据在平均值的三个标准偏差范围内。落在三个标准偏差之外的任何数据通常被认为是异常值，因为它与平均值相差很多。例如，假设您正在监控 EC2 实例的 CPU 使用率，以确保客户有良好的体验。如果您监控平均值，这可以隐藏异常值。如果您监控最大值，单个异常值可能会使结果出现偏差。使用百分位数，您可以监控 CPU 使用率的第 95 个百分位数，以检查负载异常重的实例。

某些 CloudWatch 指标支持百分位数作为统计数据。对于这些指标，您可以使用百分位数监控您的系统和应用程序，就像使用其他 CloudWatch 统计数据（平均值、最小值、最大值和总和）一样。例如，在创建警报时，可以使用百分位数作为统计函数。您可以指定最多十个小数位的百分位数（例如 p95.0123456789）。

百分位数统计数据可用于自定义指标，只要您为自定义指标发布原始、未经汇总的数据点即可。当任何指标值为负数时，百分位数统计数据不可用于指标。

CloudWatch 需要原始数据点来计算百分位数。如果您改用统计数据集发布数据，只有满足以下条件之一，才能检索此数据的百分位数统计数据：

- 统计数据集的 SampleCount 值为 1，且最小值、最大值和总和均相等。
- 最小值和最大值相等，总和等于最小值乘以 SampleCount。

以下 AWS 服务包含支持百分位数统计的指标。

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

CloudWatch 还支持切尾均值和其他性能统计数据，这些统计数据的使用与百分位数类似。有关更多信息，请参阅 CloudWatch 统计数据定义。

告警

您可使用警报 代表您自动发起操作。警报在指定的时间段内监控单个指标，并根据指标值相对于阈值的变化情况执行一项或多项指定操作。操作是一个发送到 Amazon SNS 主题或 Auto Scaling 策略的通知。您还可以将警报添加到控制面板。

告警仅为持续状态更改调用操作。CloudWatch 告警将不会调用操作，因为这些操作处于特定状态。该状态必须已改变并在指定的若干个时间段内保持不变。

创建告警时，请选择一个大于或等于指标的精度的告警监控周期。例如，对 Amazon EC2 进行的基本监控每隔 5 分钟提供一次实例指标。为基本监控指标设置警报时，选择的时间段至少应为 300 秒（5 分钟）。对 Amazon EC2 的详细监控以 1 分钟的精度提供实例指标。当为详细监控指标设置警报时，选择的时间段至少为 60 秒 (1 分钟)。

如果对高精度指标设置告警，您可以指定 10 秒或 30 秒时间段的高精度告警，也可以设置 60 秒的任意倍数时间段的定期告警。高精度告警的费用较高。有关高精度指标的更多信息，请参阅 发布 自定义指标。

有关更多信息，请参阅 使用 Amazon CloudWatch 告警 和从图表上的指标创建告警。

账单和成本

有关 CloudWatch 定价的详细信息，请参阅 Amazon CloudWatch 定价。

有关可以帮助分析账单以及优化和降低成本的信息，请参阅 CloudWatch 账单和成本。

Amazon CloudWatch 资源

下列相关资源在您使用此服务的过程中会有所帮助。

资源	描述
Amazon CloudWatch 常见问题	“常见问题解答”涵盖了开发人员对此产品提出的一些最热门的问题。
AWS 开发人员中心	这是一个帮助您入门的资源整合点，您可以在这里找到相关的文档、代码示例、发布说明和其他信息，帮助您通过 AWS 构建创新的应用程序。
AWS Management Console	此控制台让您无需进行编程即可执行 Amazon CloudWatch 和其他各种 AWS 产品的大部分功能。
Amazon CloudWatch 论坛	由开发人员组成的社群形式的论坛，开发人员可在这里讨论与 Amazon CloudWatch 有关的技术问题。
AWS Support	用于创建和管理 AWS Support 案例的中心。还包括指向其他有用资源的链接，如论坛、技术常见问题、服务运行状况和 AWS Trusted Advisor。
Amazon CloudWatch 产品信息	提供了 Amazon CloudWatch 相关信息的主要 Web 页面。
联系我们	用来查询 AWS 账单、账户、事件、滥用等信息的中心联络点。

开始设置

要使用 Amazon CloudWatch，您需要一个 AWS 账户。您的 AWS 账户允许您使用 Amazon EC2 等服务，来生成可在 CloudWatch 控制台（一种基于 Web 的点击式界面）中查看的指标。此外，您可以安装并配置 AWS 命令行界面 (CLI)。

注册 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

注册过程完成后，AWS 会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册 AWS 账户 后，请保护好您的 AWS 账户根用户，启用 AWS IAM Identity Center，并创建一个管理用户，以避免使用根用户执行日常任务。

保护您的 AWS 账户根用户

1. 选择根用户并输入您的 AWS 账户电子邮件地址，以账户拥有者身份登录 AWS Management Console。在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅《IAM 用户指南》中的为 AWS 账户根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关如何使用 IAM Identity Center 目录 作为身份源的教程，请参阅《AWS IAM Identity Center 用户指南》中的使用默认的 IAM Identity Center 目录 配置用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  要获取使用 IAM Identity Center 用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

登录 Amazon CloudWatch 控制台

登录 Amazon CloudWatch 控制台

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 如果需要，可使用导航栏将该区域更改为 AWS 资源所在的区域。
3. 即使这是您首次使用 CloudWatch 控制台，Your Metrics（您的指标）也可能已报告指标，因为您使用的 AWS 产品会免费自动将指标推送到 Amazon CloudWatch。其他服务要求您启用指标。

   如果您没有任何警报，则“Your Alarms”区段将有一个“Create Alarm”按钮。

设置 AWS CLI

您可以使用 AWS CLI 或 Amazon CloudWatch CLI 来执行 CloudWatch 命令。请注意，AWS CLI 会替换 CloudWatch CLI，我们仅将新的 CloudWatch 功能包含在此 AWS CLI 中。

有关如何安装和配置 AWS CLI 的信息，请参阅 AWS Command Line Interface 用户指南中的使用 AWS Command Line Interface 进行设置。

有关如何安装和配置 Amazon CloudWatch CLI 的信息，请参阅 Amazon CloudWatch CLI 参考中的设置 Command Line Interface。

Amazon CloudWatch 入门

访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

将显示 CloudWatch 概览主页。

概览显示以下项目且自动刷新。

- 按 AWS 服务列出的警报会显示您在账户中使用的 AWS 服务列表以及这些服务中的警报状态。旁边会显示您账户中的两个或四个警报。该数字取决于您使用的 AWS 服务数量。显示的警报是处于 ALARM 状态或最近更改状态的警报。

  这些上部区域可帮助您通过查看每项服务中的警报状态和最新更改了状态的警报，评测 AWS 服务的运行状况。这有助于您监控并快速诊断问题。
- 这些区域下方是默认控制面板（如果有）。默认控制面板是您已创建且命名为 CloudWatch-Default 的自定义控制面板。这是一种方便的方式，供您将有关自己的自定义服务或应用程序的指标添加到概览页面，或从您最想监控的 AWS 服务引入其他关键指标。

> **Note**
>
> CloudWatch 主页上的自动控制面板仅显示当前账户中的信息，即使该账户是为 CloudWatch 跨账户可观测性设置的监控账户，也是如此。有关创建自定义跨账户控制面板的信息，请参阅 CloudWatch 跨账户可观测性控制面板。

在此概述中，您可以看到包含来自多个服务的指标的跨 AWS 服务控制面板，或者将视图重点放在特定的资源组或特定 AWS 服务上。这样，您就可以将视图缩小到您感兴趣的资源子集。有关更多信息，请参阅以下部分。

查看单项服务的自动预构建控制面板

查看单项服务的自动预构建控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   此时将显示主页。
2. 在左侧导航窗格中，选择控制面板。
3. 选择自动控制面板选项卡，然后选择要查看的服务。
4. 要切换到查看此服务的警报，请在当前显示服务名称的屏幕顶部附近选中告警中、数据不足或确定复选框。
5. 当查看指标时，您可以通过多种方式专注于特定指标：

   a. 要查看任何图表中有关指标的更多详细信息，请将鼠标悬停在图表上，选择操作图标根据指标查看。

      该图表显示在新标签中，图表下方列出相关的指标。您可以自定义此图表的视图，同时更改显示的指标和资源、统计数据、时间段以及其他因素，以更好地了解当前情况。

   b. 您可以查看图表中显示的时间范围内的日志事件。这可以帮助您发现您的基础设施中发生的导致意外更改您的指标的事件。

      要查看日志事件，请将鼠标悬停报图表上，选择操作图标，然后选择 View in logs (在日志中查看)。

      该 CloudWatch Logs 视图会在新选项卡中出现，并显示您的日志组列表。要查看在原始图表中显示的时间范围内发生的其中一个日志组中的日志事件，请选择该日志组。
6. 当查看警报时，您可以通过多种方式专注于特定警报：

   - 要查看有关警报的更多详细信息，请将鼠标悬停在警报上，选择操作图标，然后选择在警报中查看。

     警报视图显示在新的标签中，其中显示您的警报列表以及有关所选警报的详细信息。要查看此警报的历史记录，请选择 History (历史记录) 选项卡。

7. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上除警报之外的项的自动刷新率，请选择刷新图标旁边的向下箭头，然后选择刷新率。还可以选择关闭自动刷新。
8. 要更改所有图表和当前显示的警报中显示的时间范围，请在屏幕顶部的时间范围旁边选择范围。要从默认显示的选项之外的多个时间范围选项进行选择，请选择自定义。
9. 要返回到跨服务控制面板，请在当前显示您专注的服务的屏幕顶部的列表中选择 Overview (概览)。

   或者，从任何视图中，您可以在屏幕顶部选择 CloudWatch，以清除所有筛选器并返回到概览页面。

查看预构建的跨服务控制面板

您可以切换到跨服务控制面板屏幕，然后与您正在使用的所有 AWS 服务的控制面板互动。CloudWatch 控制台按字母顺序显示您的控制面板，并在每个控制面板上显示一两个关键指标。

> **Note**
>
> 如果您使用五个或更多 AWS 服务，CloudWatch 控制台不会在概览屏幕上显示跨服务控制面板。

打开跨服务控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   会将您定向到概览屏幕。
2. 在概览屏幕中，选择显示 Overview（概览）的下拉菜单，然后选择 Cross service dashboard（跨服务控制面板）。

   会将您定向到跨服务控制面板屏幕。
3. （可选）如果您使用原始界面，请滚动到 Cross-service dashboard（跨服务控制面板）部分，然后选择 View Cross-service dashboard（查看跨服务控制面板）。

   会将您定向到跨服务控制面板屏幕。
4. 您可以按两种方式专注于特定服务：

a. 要查看服务的更多关键指标，请从当前显示 Cross service dashboard (跨服务控制面板) 的屏幕顶部的列表中选择其名称。或者，您可以选择服务名称旁边的 View Service dashboard (查看服务控制面板) 复选框。

   将显示针对该服务的自动控制面板，其中显示该服务的更多指标。此外，对于某些服务，服务控制面板的底部显示与该服务相关的资源。您可以选择该服务控制台中的其中一个资源，并进一步专注于该资源。

b. 要查看与服务相关的所有警报，请选择屏幕右侧该服务名称旁边的按钮。此按钮上的文本指示您在此服务中已经创建的警报数量，以及任意警报是否处于 ALARM 状态。

   显示多个警报时，具有相似设置（如维度、阈值或时间段）的多个警报可能显示在一个图中。

   然后，您可以查看有关警报的详细信息和查看警报历史记录。为此，请将鼠标悬停在警报图表上，选择操作图标，然后选择在警报中查看。

   警报视图显示在新的浏览器标签中，其中显示您的警报列表以及有关所选警报的详细信息。要查看此警报的历史记录，请选择 History (历史记录) 选项卡。

5. 您可以专注于特定资源组中的资源。为此，请从其中显示所有资源的页面顶部的列表中选择资源组。

   有关更多信息，请参阅 查看资源组的预构建控制面板。

6. 要更改所有图表和当前显示的警报中显示的时间范围，请在屏幕顶部的时间范围旁边选择需要的范围。选择自定义可从默认显示的选项之外的多个时间范围选项进行选择。

7. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上警报之外的项的自动刷新频率，请选择刷新图标旁边的向下箭头，然后选择需要的刷新率。还可以选择关闭自动刷新。

将服务从跨服务控制面板中移除

您可以防止服务的指标出现在跨服务控制面板中。这有助于您专注于跨服务控制面板以了解您最需要监控的服务。

如果您从跨服务控制面板上删除某个服务，针对该服务的告警仍然会出现在告警的视图中。

从跨服务控制面板中删除服务的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

此时将显示主页。

2. 在页面顶部，在概览下选择您要删除的服务。

   视图将改为仅显示该服务的指标。

3. 选择 Actions (操作)，然后清除 Show on cross service dashboard (在跨服务控制面板上显示) 旁边的复选框。

查看资源组的预构建控制面板

您可以将您的视图专注于显示单个资源组中的指标和警报。使用资源组，您可以使用标签来组织项目，专注于您架构的子集，或区分您的生产环境与开发环境。它们还使您能够专注于 CloudWatch 概览上的每个资源组。有关更多信息，请参阅什么是 AWS Resource Groups？。

当您专注于一个资源组时，显示的内容将更改，而只显示您在其中将资源标记为此资源组一部分的服务。最近的告警区域仅显示与资源组的资源相关联的告警。此外，如果您创建了一个名为 CloudWatch-Default-ResourceGroupName 的控制面板，它将显示在 Default dashboard (默认控制面板) 区域中。

您可以通过同时专注于单个 AWS 服务和资源组来进一步向下钻取。以下过程仅说明如何专注于一个资源组。

专注于单个资源组

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 从其中显示所有资源的页面顶部，选择一个资源组。
3. 要查看与此资源组相关的更多指标，请在屏幕底部附近选择 View cross service dashboard (查看跨服务控制面板)。

   此时将显示跨服务控制面板，同时只显示与此资源组相关的服务。对于每项服务，将显示一个或两个关键指标。

4. 要更改所有图表和当前显示的警报中显示的时间范围，对于屏幕顶部的时间范围，选择一个范围。要从默认显示的选项之外的多个时间范围选项进行选择，请选择自定义。
5. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上除警报之外的项的自动刷新率，请选择刷新图标旁边的向下箭头，然后选择刷新率。还可以选择关闭自动刷新。

6. 要返回以显示有关您账户中所有资源的信息，请在当前显示资源组名称的屏幕顶部附近，选择所有资源。

查看预构建的跨服务控制面板

您可以切换到跨服务控制面板屏幕，然后与您正在使用的所有 AWS 服务的控制面板互动。CloudWatch 控制台按字母顺序显示您的控制面板，并从每个服务上显示一两个关键指标。

> **Note**
>
> 如果您使用五个或更多 AWS 服务，CloudWatch 控制台不会在概览屏幕上显示跨服务控制面板。

打开跨服务控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   会将您定向到概览屏幕。
2. 在概览屏幕中，选择显示 Overview（概览）的下拉菜单，然后选择 Cross service dashboard（跨服务控制面板）。

   会将您定向到跨服务控制面板屏幕。
3. （可选）如果您使用原始界面，请滚动到 Cross-service dashboard（跨服务控制面板）部分，然后选择 View Cross-service dashboard（查看跨服务控制面板）。

   会将您定向到跨服务控制面板屏幕。
4. 您可以按两种方式专注于特定服务：

   a. 要查看服务的更多关键指标，请从当前显示 Cross service dashboard (跨服务控制面板) 的屏幕顶部的列表中选择其名称。或者，您可以选择服务名称旁边的 View Service dashboard (查看服务控制面板) 复选框。

      将显示针对该服务的自动控制面板，其中显示该服务的更多指标。此外，对于某些服务，服务控制面板的底部显示与该服务相关的资源。您可以选择该服务控制台中的其中一个资源，并进一步专注于该资源。

   b. 要查看与服务相关的所有警报，请选择屏幕右侧该服务名称旁边的按钮。此按钮上的文本指示您在此服务中已经创建的警报数量，以及任意警报是否处于 ALARM 状态。

显示多个警报时，具有相似设置（如维度、阈值或时间段）的多个警报可能显示在一个图中。

然后，您可以查看有关警报的详细信息和查看警报历史记录。为此，请将鼠标悬停在警报图表上，选择操作图标，然后选择在警报中查看。

警报视图显示在新的浏览器标签中，其中显示您的警报列表以及有关所选警报的详细信息。要查看此警报的历史记录，请选择 History (历史记录) 选项卡。

5. 您可以专注于特定资源组中的资源。为此，请从其中显示所有资源的页面顶部的列表中选择资源组。

   有关更多信息，请参阅 查看资源组的预构建控制面板。

6. 要更改所有图表和当前显示的警报中显示的时间范围，请在屏幕顶部的时间范围旁边选择需要的范围。选择自定义可从默认显示的选项之外的多个时间范围选项进行选择。

7. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上警报之外的项的自动刷新频率，请选择刷新图标旁边的向下箭头，然后选择需要的刷新率。还可以选择关闭自动刷新。

使服务不出现在跨服务控制面板中

您可以防止服务的指标出现在跨服务控制面板中。这有助于您专注于跨服务控制面板以了解您最需要监控的服务。

如果您从跨服务控制面板上删除某个服务，针对该服务的告警仍然会出现在告警的视图中。

从跨服务控制面板中删除服务的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   此时将显示主页。

2. 在页面顶部，在概览下选择您要删除的服务。

   视图将改为仅显示该服务的指标。

3. 选择 Actions (操作)，然后清除 Show on cross service dashboard (在跨服务控制面板上显示) 旁边的复选框。

查看单项 AWS 服务的预构建控制面板

在 CloudWatch 主页上，您可以让视图专注于单个 AWS 服务。您可以通过同时专注于单个 AWS 服务和资源组来进一步向下钻取。以下过程显示如何仅专注于一个 AWS 服务。

专注于单个服务

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

   此时将显示主页。
2. 对于概述（概述当前显示在下拉菜单中），选择服务控制面板。
3. 选择要专注的服务。

   视图将改为显示所选服务的关键指标的图表。
4. 要切换到查看此服务的警报，请在当前显示服务名称的屏幕顶部附近选中告警中、数据不足或确定复选框。
5. 当查看指标时，您可以通过多种方式专注于特定指标：

   a. 要查看任何图表中有关指标的更多详细信息，请将鼠标悬停在图表上，选择操作图标根据指标查看。

      该图表显示在新标签中，图表下方列出相关的指标。您可以自定义此图表的视图，同时更改显示的指标和资源、统计数据、时间段以及其他因素，以更好地了解当前情况。

   b. 您可以查看图表中显示的时间范围内的日志事件。这可以帮助您发现您的基础设施中发生的导致意外更改您的指标的事件。

      要查看日志事件，请将鼠标悬停报图表上，选择操作图标，然后选择 View in logs (在日志中查看)。

      该 CloudWatch Logs 视图会在新选项卡中出现，并显示您的日志组列表。要查看在原始图表中显示的时间范围内发生的其中一个日志组中的日志事件，请选择该日志组。
6. 当查看警报时，您可以通过多种方式专注于特定警报：

   - 要查看有关警报的更多详细信息，请将鼠标悬停在警报上，选择操作图标，然后选择在警报中查看。

     警报视图显示在新的标签中，其中显示您的警报列表以及有关所选警报的详细信息。要查看此警报的历史记录，请选择 History (历史记录) 选项卡。

7. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上除警报之外的项的自动刷新率，请选择刷新图标旁边的向下箭头，然后选择刷新率。还可以选择关闭自动刷新。
8. 要更改所有图表和当前显示的警报中显示的时间范围，请在屏幕顶部的时间范围旁边选择范围。要从默认显示的选项之外的多个时间范围选项进行选择，请选择自定义。
9. 要返回到跨服务控制面板，请在当前显示您专注的服务的屏幕顶部的列表中选择 Overview (概览)。

   或者，从任何视图中，您可以在屏幕顶部选择 CloudWatch，以清除所有筛选器并返回到概览页面。

查看资源组的预构建控制面板

您可以将您的视图专注于显示单个资源组中的指标和警报。使用资源组，您可以使用标签来组织项目，专注于您架构的子集，或区分您的生产环境与开发环境。它们还使您能够专注于 CloudWatch 概览上的每个资源组。有关更多信息，请参阅什么是 AWS Resource Groups？。

当您专注于一个资源组时，显示的内容将更改，而只显示您在其中将资源标记为此资源组一部分的服务。最近的告警区域仅显示与资源组的资源相关联的告警。此外，如果您创建了一个名为 CloudWatch-Default-ResourceGroupName 的控制面板，它将显示在 Default dashboard (默认控制面板) 区域中。

您可以通过同时专注于单个 AWS 服务和资源组来进一步向下钻取。以下过程显示如何仅专注于一个资源组。

专注于单个资源组

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 从其中显示所有资源的页面顶部，选择一个资源组。
3. 要查看与此资源组相关的更多指标，请在屏幕底部附近选择 View cross service dashboard (查看跨服务控制面板)。

   此时将显示跨服务控制面板，同时只显示与此资源组相关的服务。对于每项服务，将显示一个或两个关键指标。
4. 要更改所有图表和当前显示的警报中显示的时间范围，对于屏幕顶部的时间范围，选择一个范围。要从默认显示的选项之外的多个时间范围选项进行选择，请选择自定义。

5. 警报始终一分钟刷新一次。要刷新该视图，请选择屏幕右上角的刷新图标（双曲线箭头）。要更改屏幕上除警报之外的项的自动刷新率，请选择刷新图标旁边的向下箭头，然后选择刷新率。还可以选择关闭自动刷新。
6. 要返回以显示有关您账户中所有资源的信息，请在当前显示资源组名称的屏幕顶部附近，选择所有资源。

CloudWatch 账单和成本

本部分介绍了 Amazon CloudWatch 功能如何产生成本。它还提供了可以帮助您分析、优化和降低 CloudWatch 成本的方法。在本部分中，我们在描述 CloudWatch 功能时有时会提及定价。有关定价的信息，请参阅 Amazon CloudWatch 定价。

主题

- 使用 Cost Explorer 分析 CloudWatch 成本和使用数据
- 使用 AWS 成本和使用情况报告 和 Athena 分析 CloudWatch 成本和使用数据
- 优化和降低成本的最佳实践

使用 Cost Explorer 分析 CloudWatch 成本和使用数据

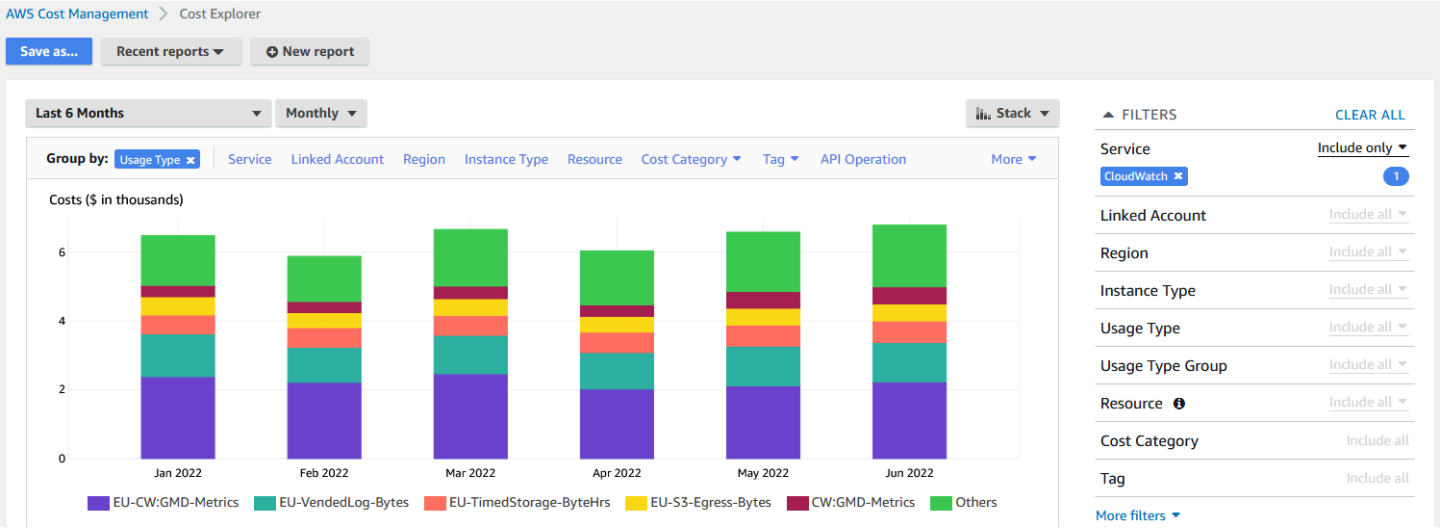
使用 AWS Cost Explorer，您可以可视化和分析 AWS 服务 一段时间内的成本和使用数据，包括 CloudWatch。有关更多信息，请参阅 AWS Cost Explorer 入门。

以下程序介绍如何使用 Cost Explorer 可视化和分析 CloudWatch 成本和使用数据。

可视化和分析 CloudWatch 成本和使用数据

1. 在 https://console.aws.amazon.com/cost-management/home#/custom 中登录 Cost Explorer 控制台。
2. 在 FILTERS（筛选条件）下，对于Service（服务），选择 CloudWatch。
3. 对于 Group by（分组依据），选择 Usage Type（使用类型）。您还可以按其他类别对结果进行分组，例如以下类别：
   - API Operation （API 操作）– 查看哪些 API 操作产生的成本最高。
   - Region （区域）– 查看哪些地区产生的成本最高。

下图显示了 CloudWatch 功能在六个月内产生的成本示例。



要了解哪些 CloudWatch 功能产生的费用最高，请查看 UsageType 的值。例如，EU-CW:GMD-Metrics 表示 CloudWatch 批量 API 请求产生的成本。

> **Note**
>
> UsageType 的字符串匹配特定的功能和区域。例如，EU-CW:GMD-Metrics（EU）的第一部分匹配欧洲（爱尔兰）区域，EU-CW:GMD-Metrics（GMD-Metrics）的第二部分匹配 CloudWatch 批量 API 请求。
>
> UsageType 的整个字符串可以格式化如下：<Region>-CW:<Feature> 或 <Region>-<Feature>。
>
> 一些 CloudWatch 功能（例如日志和警报）也使用Global区域来识别免费套餐的用量。例如，Global-DataScanned-Bytes 表示免费的 CloudWatch Logs 数据摄取用量。
>
> 为了增强可读性，本文档的表格中的 UsageType 字符串已缩短为字符串后缀。例如，EU-CW:GMD-Metrics 缩短为 GMD-Metrics。

下表包含每个 CloudWatch 功能的名称，列出了每个子功能的名称，并列出了 UsageType 的字符串。

CloudWatch 功能	CloudWatch 子功能	**UsageType**
CloudWatch metrics（CloudWatch 指标）	自定义指标	MetricMonitorUsage
	详细监控	MetricMonitorUsage

CloudWatch 功能	CloudWatch 子功能	**UsageType**
	嵌入式指标	MetricMonitorUsage
CloudWatch API 请求	API 请求	Requests
	批量（获取）	GMD-Metrics
	Contributor Insights	GIRR-Metrics
	位图图像快照	GMWI-Metrics
CloudWatch metric streams（CloudWatch 指标流）	指标流	MetricStreamUsage
CloudWatch dashboards（CloudWatch 控制面板）	包含 50 个或更少指标的控制面板	DashboardsUsageHour-Basic
	包含超过 50 个指标的控制面板	DashboardsUsageHour
CloudWatch alarms（CloudWatch 告警）	标准（指标告警）	AlarmMonitorUsage
	高分辨率（指标告警）	HighResAlarmMonitorUsage
	Metrics Insights 查询告警	MetricInsightAlarmUsage
	复合（聚合告警）	CompositeAlarmMonitorUsage

CloudWatch 功能	CloudWatch 子功能	**UsageType**
CloudWatch Application Signals	Application Signals	`Application-Signals`
CloudWatch custom logs（CloudWatch 自定义日志）	收集（标准日志类的数据摄取）	`DataProcessing-Bytes`
	收集（不频繁访问日志类的数据摄取）	`DataProcessingIA-Bytes`
	分析（查询）	`DataScanned-Bytes`
	分析（Live Tail）	`Logs-LiveTail`
	存储（存档）	`TimedStorage-ByteHrs`
	检测和屏蔽（数据保护）	`DataProtection-Bytes`
CloudWatch 出售的日志	传送（Amazon CloudWatch Logs 标准日志类）	`VendedLog-Bytes`
	传送（CloudWatch Logs 不频繁访问日志类）	`VendedLogIA-Bytes`
	传输（Amazon S3）	`S3-Egress-Bytes`
	以 Parquet 格式的传送（Amazon S3）	`S3-Egress-InputBytes`
	传输（Amazon Data Firehose）	`FH-Egress-Bytes`

CloudWatch 功能	CloudWatch 子功能	**UsageType**
Contributor Insights	CloudWatch Logs（规则）	`ContributorInsightRules`
	CloudWatch Logs（事件）	`ContributorInsightEvents`
	Amazon DynamoDB（规则）	`ContributorRulesManaged`
	DynamoDB（事件）	`ContributorEventsManaged`
Canaries (Synthetics)	Run	`Canary-runs`
Evidently	事件	`Evidently-event`
	分析单位	`Evidently-eau`
RUM	事件	`RUM-event`

使用 AWS 成本和使用情况报告 和 Athena 分析 CloudWatch 成本和使用数据

分析 CloudWatch 成本和使用数据的另一种方法是使用 AWS 成本和使用情况报告 和 Amazon Athena。AWS 成本和使用情况报告 包含一组全面的成本和使用数据。您可以创建跟踪费用和使用量的报告，并将这些报告发布到您选择的 S3 存储桶。您还可以从 S3 存储桶中下载和删除报告。有关更多信息，请参阅《AWS 成本和使用情况报告 用户指南》中的 什么是 AWS 成本和使用情况报告？。

> **Note**
>
> 使用 AWS 成本和使用情况报告 无任何费用。您只需在将报告发布到 Amazon Simple Storage Service（Amazon S3）时支付存储费用。有关更多信息，请参阅《AWS 成本和使用情况报告 用户指南》中的 限额和限制。

Athena 是一项查询服务，您可以将其与 AWS 成本和使用情况报告 结合使用来分析成本和使用数据。您可以在 S3 存储桶中查询报告，而无需先下载它们。有关更多信息，请参阅《Amazon Athena 用户指南》中的 什么是 Amazon Athena？。有关更多信息，请参阅《Amazon Athena 用户指南》中的什么是 Amazon Athena？。有关定价的信息，请参阅 Amazon Athena 定价。

以下程序描述了启用 AWS 成本和使用情况报告 并将该服务与 Athena 集成的过程。该程序包含两个可用于分析 CloudWatch 成本和使用数据的示例查询。

> **Note**
>
> 您可以使用本文档中的任何示例查询。本文档中的所有示例查询都对应一个名为 costandusagereport 的数据库，并显示 2022 年 4 月和 2022 年的结果。您可以更改此信息。但是，在运行查询之前，请确保数据库的名称与查询中的数据库名称匹配。

使用 AWS 成本和使用情况报告 和 Athena 分析成本和使用数据

1. 启用 AWS 成本和使用情况报告。有关更多信息，请参阅《AWS 成本和使用情况报告 用户指南》中的 创建成本和使用情况报告。

   > **Tip**
   >
   > 创建报告时，确保选择 Include resource IDs（包含资源 ID）。否则，您的报告将不会包含列 `line_item_resource_id`。此行可帮助您在分析成本和使用数据时进一步确定成本。

2. 将 AWS 成本和使用情况报告 与 Athena 集成。有关更多信息，请参阅《AWS 成本和使用情况报告 用户指南》中的 使用 AWS CloudFormation 模板设置 Athena。
3. 查询您的成本和使用情况报告。

Example: Athena query（示例：Athena 查询）

您可以使用以下查询显示在给定月份中哪些 CloudWatch 功能产生的成本最高。

```
SELECT
CASE
-- Metrics
WHEN line_item_usage_type LIKE '%%MetricMonitorUsage%%' THEN 'Metrics (Custom, Detailed
 monitoring management portal EMF)'
WHEN line_item_usage_type LIKE '%%Requests%%' THEN 'Metrics (API Requests)'
WHEN line_item_usage_type LIKE '%%GMD-Metrics%%' THEN 'Metrics (Bulk API Requests)'
WHEN line_item_usage_type LIKE '%%MetricStreamUsage%%' THEN 'Metric Streams'
-- Dashboard
WHEN line_item_usage_type LIKE '%%DashboardsUsageHour%%' THEN 'Dashboards'
-- Alarms
WHEN line_item_usage_type LIKE '%%AlarmMonitorUsage%%' THEN 'Alarms (Standard)'
WHEN line_item_usage_type LIKE '%%HighResAlarmMonitorUsage%%' THEN 'Alarms (High
 Resolution)'
WHEN line_item_usage_type LIKE '%%MetricInsightAlarmUsage%%' THEN 'Alarms (Metrics
 Insights)'
WHEN line_item_usage_type LIKE '%%CompositeAlarmMonitorUsage%%' THEN 'Alarms
 (Composite)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessing-Bytes%%' THEN 'Logs (Collect - Data
 Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProcessingIA-Bytes%%' THEN 'Infrequent Access
 Logs (Collect - Data Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProtection-Bytes%%' THEN 'Logs (Data Protection -
 Detect and Mask)'
WHEN line_item_usage_type LIKE '%%TimedStorage-ByteHrs%%' THEN 'Logs (Storage -
 Archival)'
WHEN line_item_usage_type LIKE '%%DataScanned-Bytes%%' THEN 'Logs (Analyze - Logs
 Insights queries)'
WHEN line_item_usage_type LIKE '%%Logs-LiveTail%%' THEN 'Logs (Analyze - Logs Live
 Tail)'
-- Vended Logs
WHEN line_item_usage_type LIKE '%%VendedLog-Bytes%%' THEN 'Vended Logs (Delivered to
 CW)'
WHEN line_item_usage_type LIKE '%%VendedLogIA-Bytes%%' THEN 'Vended Infrequent Access
 Logs (Delivered to CW)'
WHEN line_item_usage_type LIKE '%%FH-Egress-Bytes%%' THEN 'Vended Logs (Delivered to
 Data Firehose)'
WHEN (line_item_usage_type LIKE '%%S3-Egress-Bytes%%') THEN 'Vended Logs (Delivered to
 S3)'
-- Other
WHEN line_item_usage_type LIKE '%%Application-Signals%%' THEN 'Application Signals'
```

```
WHEN line_item_usage_type LIKE '%%Canary-runs%%' THEN 'Synthetics'
WHEN line_item_usage_type LIKE '%%Evidently%%' THEN 'Evidently'
WHEN line_item_usage_type LIKE '%%RUM-event%%' THEN 'RUM'
ELSE 'Others'
END AS UsageType,
-- REGEXP_EXTRACT(line_item_resource_id,'^(?:.+?:){5}(.+)$',1) as ResourceID,
-- SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
GROUP BY
1
ORDER BY
TotalSpend DESC,
UsageType;
```

Example: Athena query（示例：Athena 查询）

您可以使用以下查询显示 UsageType 和 Operation 的结果。这向您展示了 CloudWatch 功能是如何产生成本的。结果还显示了 UsageQuantity 和 TotalSpend 的值，以便您可以查看总使用成本。

> **Tip**
>
> 有关 UsageType 的更多信息，将以下一行添加到此查询：
> `line_item_line_item_description`
> 此行创建一个名为 Description（描述）的列。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
```

```
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
```

优化和降低成本的最佳实践

CloudWatch 指标

许多 AWS 服务，例如 Amazon Elastic Compute Cloud（Amazon EC2）、Amazon S3 和 Amazon Data Firehose 等会自动向 CloudWatch 免费发送指标。但是，按以下类别分组的指标可能会产生额外费用：

- Custom metrics, detailed monitoring, and embedded metrics（自定义指标、详细监控和嵌入式指标）
- API 请求
- 指标流

有关更多信息，请参阅使用 Amazon CloudWatch 指标。

自定义指标、详细监控和嵌入式指标

自定义指标

您可以创建自定义指标以按任何顺序和任何速率组织数据点。

所有自定义指标均按小时按比例分配。它们只有在发送到 CloudWatch 时才会计量。有关指标定价的信息，请参阅 Amazon CloudWatch 定价。

下表列出了 CloudWatch 指标的相关子功能的名称。该表包含 UsageType 和 Operation 的字符串，它可以帮助您分析和确定与指标相关的成本。

> **Note**
>
> 要在使用 Athena 查询成本和使用数据时获得下表中列出的指标的更多详细信息，请将 Operation 的字符串与为 line_item_operation 显示的结果匹配。

CloudWatch 子功能	**UsageType**	**Operation**	用途
自定义指标	MetricMonitorUsage	MetricStorage	自定义指标
详细监控	MetricMonitorUsage	MetricStorage:AWS/*{Service}*	详细监控
嵌入式指标	MetricMonitorUsage	MetricStorage:AWS/Logs-EMF	日志嵌入式指标
日志筛选条件	MetricMonitorUsage	MetricStorage:AWS/CloudWatchLogs	日志组指标筛选条件

详细监控

CloudWatch 具有两种监控类型：

- 基本监控

  基本监控是免费的，并自动为支持该功能的所有 AWS 服务 启用。
- 详细监控

详细监控会产生成本，并添加不同的增强功能，具体取决于 AWS 服务。对于支持详细监控的每个 AWS 服务，您可以选择是否为该服务启用它。有关更多信息，请参阅基本和详细监控。

> **Note**
>
> 其他 AWS 服务 支持详细监控，可能使用其他名称引用此功能。例如，对 Amazon S3 的详细监控称为request metrics（请求指标）。

与自定义指标类似，详细监控按小时按比例分配，并且仅在将数据发送到 CloudWatch 时计量。详细监控根据发送到 CloudWatch 的指标数量生成成本。为降低成本，请仅在必要时启用详细监控。有关详细监控定价方式的信息，请参阅 Amazon CloudWatch 定价。

Example: Athena query（示例：Athena 查询）

您可以使用以下查询显示哪些 EC2 实例已启用详细监控。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation='MetricStorage:AWS/EC2'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation,
line_item_line_item_description
```

```
ORDER BY line_item_operation
```

嵌入式指标

借助 CloudWatch 嵌入式指标格式，您可以将应用程序数据作为日志数据提取，以便生成可操作的指标。有关更多信息，请参阅采用 CloudWatch 嵌入式指标格式摄取高基数日志并生成指标。

嵌入式指标会根据摄取的日志数量、存档的日志数和生成的自定义指标的数量产生成本。

下表列出了 CloudWatch 嵌入式指标格式的相关子功能的名称。该表包含 `UsageType` 和 `Operation` 的字符串，它可以帮助您分析和确定成本。

CloudWatch 子功能	**`UsageType`**	**`Operation`**	用途
自定义指标	`MetricMonitorUsage`	`MetricStorage:AWS/Logs-EMF`	日志嵌入式指标
日志摄取	`DataProcessing-Bytes`	`PutLogEvents`	将一批日志事件上传到指定的日志组或日志流
日志存档	`TimedStorage-ByteHrs`	`HourlyStorageMetering`	在 CloudWatch Logs 中存储每小时日志和每字节日志

要分析成本，请结合使用 AWS 成本和使用情况报告 和 Athena，以便您可以确定哪些指标正在产生成本，并确定成本是如何产生的。

要充分利用 CloudWatch 嵌入式指标格式产生的成本，请避免基于高基数维度创建指标。这样，CloudWatch 就不会为每个唯一的维度组合创建自定义指标。有关更多信息，请参阅维度。

如果您正在使用 CloudWatch Container Insights 来利用嵌入式指标格式，则可以使用 AWS Distro for Open Telemetry 作为最大限度地利用指标相关成本的替代方案。使用 Container Insights，您可以从容器化应用程序和微服务中收集、聚合和汇总指标与日志。启用 Container Insights 后，CloudWatch 代理会将您的日志发送到 CloudWatch，这样它就可以使用这些日志生成嵌入式指标。但是，CloudWatch 代理仅向 CloudWatch 发送固定数量的指标，并且您需要为所有可用指标（包括未使用的指标）付费。使用 AWS Distro for Open Telemetry，您可以配置和自定义将哪些指标和维

度发送到 CloudWatch。这有助于您减少 Container Insights 生成的数据量和降低成本。有关更多信息，请参阅以下资源：

- 使用 Container Insights
- AWS Distro for Open Telemetry

API 请求

CloudWatch 具有以下类型的 API 请求：

- API requests（API 请求）
- Bulk (Get)（批量（获取））
- Contributor Insights
- Bitmap image snapshot（位图图像快照）

API 请求会根据请求类型和请求的指标数量生成成本。

下表列出了 API 请求的类型并包含 `UsageType` 和 `Operation` 的字符串，它可以帮助您分析和确定与 API 相关的成本。

API 请求类型	**`UsageType`**	**`Operation`**	用途
API 请求	`Requests`	`GetMetric Statistics`	检索指定指标的统计数据
	`Requests`	`ListMetrics`	列出指定的指标
	`Requests`	`PutMetricData`	将指标数据点发布到 CloudWatch
	`Requests`	`GetDashboard`	显示指定控制面板的详细信息
	`Requests`	`ListDashboards`	列出您账户中的控制面板
	`Requests`	`PutDashboard`	创建或更新控制面板

API 请求类型	**UsageType**	**Operation**	用途
	Requests	DeleteDashboards	删除所有指定的控制面板
批量（获取）	GMD-Metrics	GetMetricData	检索 CloudWatch 指标值
Contributor Insights	GIRR-Metrics	GetInsightRuleReport	返回由 Contributor Insights 规则收集的时间序列数据
位图图像快照	GMWI-Metrics	GetMetricWidgetImage	将一个或多个 CloudWatch 指标的图表快照作为位图图像检索

要分析成本，请使用 Cost Explorer，并按 API Operation（API 操作）对结果进行分组。

API 请求的成本各不相同，当您的 API 调用次数超过根据 AWS 免费套餐限制提供给您的 API 调用次数时，会产生费用。

> **Note**
>
> GetMetricData 和 GetMetricWidgetImage 不包含在 AWS 免费套餐限制下。有关更多信息，请参阅《AWS Billing 用户指南》中的 使用 AWS 免费套餐。

通常会增加成本的 API 请求是 Put 和 Get 请求。

PutMetricData

PutMetricData 每次调用时都会产生成本，并且根据使用案例的不同，可能会产生巨额成本。有关更多信息，请参阅 Amazon CloudWatch API 参考中的 PutMetricData。

要最大限度地利用 PutMetricData 产生的成本，将更多数据批量发送到您的 API 调用中。根据您的使用案例，考虑使用 CloudWatch Logs 或 CloudWatch 嵌入式指标格式来注入指标数据。有关更多信息，请参阅以下资源：

- Amazon CloudWatch Logs 用户指南中的什么是 Amazon CloudWatch Logs？
- 采用 CloudWatch 嵌入式指标格式摄取高基数日志并生成指标
- 利用 Amazon CloudWatch 嵌入式自定义指标降低成本并关注我们的客户

GetMetricData

GetMetricData 也可能会产生巨大的成本。提高成本的常见使用案例涉及第三方监控工具，这些工具会提取数据以生成见解。有关更多信息，请参阅 Amazon CloudWatch API 参考中的 GetMetricData。

要降低 GetMetricData 产生的成本，请考虑只提取受监控和使用的数据，或者考虑减少提取数据的频率。根据您的使用案例，您可能会考虑使用指标流，而不是 GetMetricData，这样您就可以以更低的成本将数据近乎实时地推送给第三方。有关更多信息，请参阅以下资源：

- 使用指标流
- CloudWatch 指标流 - 将 AWS 指标实时发送给合作伙伴和您的应用程序

GetMetricStatistics

根据您的使用案例，您可能会考虑使用 GetMetricStatistics 而不是 GetMetricData。使用 GetMetricData，您可以快速大规模地检索数据。但是，GetMetricStatistics 包含在 AWS 免费套餐限制下，最多一百万个 API 请求，如果您不需要在每次调用时检索那么多的指标和数据点，这可以帮助您降低成本。有关更多信息，请参阅以下资源：

- Amazon CloudWatch API 参考中的 GetMetricStatistics
- 我应该使用 GetMetricData 还是 GetMetricStatistics？

> **Note**
>
> 外部调用者进行 API 调用。对于 CloudTrail 数据事件支持的 API（例如 GetMetricData 和 GetMetricWidgetImage），您可以使用 CloudTrail 来识别排名靠前的 CloudWatch API 调用者，并有可能减少或识别意外调用。有关更多信息，请参阅如何使用 CloudTrail 分析您的 CloudWatch API 用量。对于 CloudTrail 不支持的其他 CloudWatch API，您可以向 CloudWatch 团队提出技术支持请求并询问有关这些 API 的信息。有关创建技术支持请求的信息，请参阅如何从 AWS 获得技术支持？

CloudWatch metric streams（CloudWatch 指标流）

借助 CloudWatch 指标流，您可以将指标持续发送到 AWS 目标和第三方服务提供商目标。

指标流根据指标更新的数量产生成本。指标更新始终包括以下统计数据的值：

- `Minimum`
- `Maximum`
- `Sample Count`
- `Sum`

有关更多信息，请参阅可以流式传输的统计数据。

要分析 CloudWatch 指标流产生的成本，请结合使用 AWS 成本和使用情况报告 与 Athena。通过此方式，您可以确定哪些指标正在产生成本，并确定成本是如何产生的。

Example: Athena query（示例：Athena 查询）

您可以使用以下查询通过 Amazon 资源名称（ARN）跟踪哪些指标流产生成本。

```
SELECT
SPLIT_PART(line_item_resource_id,'/',2) AS "Stream Name",
line_item_resource_id as ARN,
SUM(CAST(line_item_unblended_cost AS decimal(16,2))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
AND line_item_usage_type LIKE '%%MetricStreamUsage%%'
GROUP BY line_item_resource_id
ORDER BY TotalSpend DESC
```

要降低 CloudWatch 指标流产生的成本，请仅流式传输能为您带来业务价值的指标。您还可以停止或暂停任何未使用的指标流。

CloudWatch 告警

通过 CloudWatch 告警，您可以创建基于单个指标的告警、基于 Metrics Insights 查询的告警以及监视其他告警的复合告警。

> **Note**
>
> 指标和复合告警的成本按小时按比例分配。只有当您的告警存在时，您才会产生告警成本。为了优化成本，请勿遗留配置错误的警报或低值警报。为此，您可以自动清理不再需要的 CloudWatch 警报。有关更多信息，请参阅 Automating Amazon CloudWatch Alarm Cleanup at Scale

Metric alarms（指标告警）

指标告警具有以下分辨率设置：

- Standard（标准）（每 60 秒评估一次）
- High resolution（高分辨率）（每 10 秒评估一次）

创建指标告警时，成本基于告警的分辨率设置和告警引用的指标数。例如，引用单项指标的指标告警，每小时产生一个告警指标成本。有关更多信息，请参阅使用 Amazon CloudWatch 告警。

如果创建的指标告警包含引用多个指标的指标数学表达式，则在指标数学表达式中引用的每个告警指标都会产生成本。有关如何创建包含指标数学表达式的指标告警的信息，请参阅根据指标数学表达式创建 CloudWatch 告警。

如果创建异常检测告警，其中告警会分析过去的指标数据以创建预期值模型，则告警中引用的每个警报指标加上两个额外的指标（分别用于异常检测模型创建的上下限指标）都会产生成本。有关如何创建异常检测告警的信息，请参阅根据异常检测创建 CloudWatch 告警。

Metrics Insights query alarms（Metrics Insights 查询告警）

Metrics Insights 查询告警是一种特定类型的指标告警，仅在标准分辨率下可用（每 60 秒评估一次）。

创建 Metrics Insights 查询告警时，您的成本取决于告警引用的查询所分析的指标数量。例如，如果 Metrics Insights 查询告警引用了筛选条件匹配十个指标的查询，则每小时会产生十个指标的分析成本。有关更多信息，请参阅 Amazon CloudWatch 定价上的定价示例。

如果您创建的告警包含 Metrics Insights 查询和指标数学表达式，则该告警将作为 Metrics Insights 查询告警进行报告。如果您的告警包含一个指标数学表达式，该表达式除了引用 Metrics Insights 查询所分析的指标外还引用了其他指标，则该指标数学表达式中引用的每个告警指标都会产生额外的成本。有关如何创建包含指标数学表达式的指标告警的信息，请参阅根据指标数学表达式创建 CloudWatch 告警。

Composite alarms（复合告警）

复合告警包含规则表达式，用于指定应如何评估其他告警的状态以确定其自身的状态。无论评估多少其他告警，复合告警每小时都会产生标准成本。在规则表达式中引用的复合告警会产生单独的成本。有关更多信息，请参阅创建复合告警。

告警使用类型

下表列出了 CloudWatch 告警的相关子功能的名称。该表包含 `UsageType` 的字符串，它可以帮助您分析和确定与告警相关的成本。

CloudWatch 子功能	**`UsageType`**
标准指标告警	`AlarmMonitorUsage`
高分辨率指标告警	`HighResAlarmMonitorUsage`
Metrics Insights 查询告警	`MetricInsightAlarmUsage`
复合告警	`CompositeAlarmMonitorUsage`

降低告警成本

要优化聚合了四个或更多指标的指标数学告警所产生的成本，您可以在将数据发送到 CloudWatch 之前聚合数据。这样，您可以创建单个指标的告警，而不是为多个指标聚合数据的告警。有关更多信息，请参阅发布自定义指标。

为了优化 Metrics Insights 查询告警产生的成本，您可以确保用于查询的筛选条件仅匹配您要监控的指标。

降低成本的最佳方法是删除所有不必要或未使用的告警。例如，您可以删除评估不再存在的 AWS 资源发出的指标的告警。

示例：使用 ***`DescribeAlarms`*** 检查处于 ***`INSUFFICIENT_DATA`*** 状态的告警

如果删除资源，但不删除该资源发出的指标告警，则告警仍然会存在，通常会进入 INSUFFICIENT_DATA 状态。要检查是否有处于 INSUFFICIENT_DATA 状态的告警，使用以下 AWS Command Line Interface（AWS CLI）命令。

```
$ aws cloudwatch describe-alarms –state-value INSUFFICIENT_DATA
```

有关更多信息，请参阅 大规模自动化 Amazon CloudWatch 警报清除。

其他降低成本的方法如下：

- 确保为正确的指标创建告警。
- 确保在您不工作的区域中未启用任何告警。
- 请记住，尽管复合告警降低了噪音，但也会产生额外的成本。
- 在决定是创建标准告警还是高分辨率告警时，请考虑您的使用案例以及每种告警类型带来的价值。

CloudWatch Logs

Amazon CloudWatch Logs 具有以下日志类型：

- Custom logs（自定义日志）（您为应用程序创建的日志）
- Vended logs（已出售日志）（其他 AWS 服务，例如 Amazon Virtual Private Cloud（Amazon VPC）和 Amazon Route 53，代表您创建的日志）

有关已出售日志的更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 启用特定 AWS 服务中的日志记录。

自定义日志和已出售日志会根据收集、存储和分析的日志数量产生成本。另外，已出售日志会针对到 Amazon S3 和 Firehose 的传输产生成本。

下表列出了 CloudWatch Logs 功能的名称和相关子功能的名称。该表包含 UsageType 和 Operation 的字符串，它可以帮助您分析和确定与日志相关的成本。

CloudWatch Logs 功能	CloudWatch Logs 子功能	**UsageType**	**Operation**	用途
Custom logs（自定义日志）	收集（标准日志类的数据摄取）	DataProcessing-Bytes	PutLogEvents	将一批日志上传到标准类日志

CloudWatch Logs 功能	CloudWatch Logs 子功能	**UsageType**	**Operation**	用途
				组中的特定日志流。
	收集（不频繁访问日志类的数据摄取）	DataProcessingIA-Bytes	PutLogEvents	将一批日志上传到不频繁访问日志组中的特定日志流。
	检测和屏蔽（数据保护）	DataProtection-Bytes	PutLogEvents	检测和屏蔽日志事件中的受保护数据。
	存储（存档）	TimedStorage-ByteHrs	HourlyStorageMetering	在 CloudWatch Logs 中存储每小时日志和每字节日志。
	分析（Logs Insights 查询）	DataScanned-Bytes	StartQuery	CloudWatch Logs Insights 查询所扫描的日志数据
	分析（Logs Live Tail）	Logs-LiveTail	StartLiveTail	在 CloudWatch Logs Live Tail 会话期间分析的日志
Vended logs（已出售日志）	交付（CloudWatch Logs 标准日志类）	VendedLog-Bytes	PutLogEvents	将一批日志上传到标准日志类中的特定日志流。
	传送（CloudWatch Logs 不频繁访问日志类）	VendedLogIA-Bytes	PutLogEvents	将一批日志上传到不频繁访问日志类中的特定日志流。

CloudWatch Logs 功能	CloudWatch Logs 子功能	**UsageType**	**Operation**	用途
	传输（Amazon S3）	S3-Egress-Bytes	LogDelivery	将一批日志上传到特定的 S3 存储桶
	以 Parquet 格式的传送（Amazon S3）	S3-Egress-InputBytes	ParquetConversion	对交付到 Amazon S3 的日志执行 Parquet 转换
	传输（Firehose）	FH-Egress-Bytes	LogDelivery	将一批出售的日志上传到 Amazon Data Firehose

要分析成本，请结合使用 AWS 成本和使用情况报告 和 Athena，以便您可以确定哪些日志正在产生成本，并确定成本是如何产生的。

Example: Athena query（示例：Athena 查询）

您可以使用以下查询按资源 ID 跟踪哪些日志会产生成本。

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_resource_id AS ResourceID,
line_item_usage_type AS UsageType,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
```

```
AND line_item_operation IN
 ('PutLogEvents','HourlyStorageMetering','StartQuery','LogDelivery','StartLiveTail','ParquetCon
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
ORDER BY
TotalSpend DESC
```

要充分利用 CloudWatch Logs 产生的成本，请考虑以下几点：

- 仅记录能为您带来业务价值的事件。这有助于您减少摄取成本。
- 更改日志保留设置，以减少存储成本。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 更改 CloudWatch Logs 中的日志数据留存。
- 运行 CloudWatch Logs Insights 自动保存在历史记录中的查询。这样，您产生的分析成本就会降低。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 查看正在运行的查询或查询历史记录。
- 使用 CloudWatch 代理收集系统日志和应用程序日志并将其发送到 CloudWatch。这样，您就可以仅收集符合条件的日志事件。有关更多信息，请参阅 Amazon CloudWatch 代理增加了对日志筛选表达式的支持。

要降低已出售日志的成本，请考虑您的使用案例，然后确定应将日志发送到 CloudWatch 还是 Amazon S3。有关的更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 发送到 Amazon S3 的日志。

> **Tip**
>
> 如果您想使用指标筛选条件、订阅筛选条件、CloudWatch Logs Insights 和 Contributor Insights，请将已出售日志发送到 CloudWatch。
> 或者，如果您正在使用 VPC 流日志并将其用于审计和合规性目的，请将已出售日志发送到 Amazon S3。
> 有关如何跟踪将 VPC 流日志发布到 S3 存储桶时产生的费用的信息，请参阅使用 AWS 成本和使用情况报告 和成本分配标签了解 Amazon S3 中的 VPC 流日志数据摄取。

有关如何充分利用 CloudWatch Logs 产生的成本的更多信息，请参阅哪个日志组导致我的 CloudWatch Logs 账单突然增加？。

使用 Amazon CloudWatch 控制面板

Amazon CloudWatch 控制面板是 CloudWatch 控制台中的可自定义主页，可用于在单个视图中监控资源，即便是分布到不同区域的资源，也能对其进行监控。您可以使用 CloudWatch 控制面板创建 AWS 资源的指标和告警的自定义视图。

利用控制面板，您可以创建以下各项：

- 所选指标和告警的单一视图，用于帮助您跨一个或多个区域评估资源和应用程序的运行状况。您可以在每个图表上选择用于每个指标的颜色，以便轻松地跨多个图表跟踪同一指标。
- 一个操作手册，为团队成员提供有关如何对操作事件期间发生的特定事故做出响应的指南。
- 关键资源与应用程序测量的公共视图，团队成员可以共享该视图，以便在操作事件期间加快通信流。

如果您有多个 AWS 账户，则可以设置 CloudWatch 跨账户可观测性，然后在监控账户中创建丰富的跨账户控制面板。这些控制面板可能包含来自源账户的指标图表，以及具有源账户日志组查询功能的 CloudWatch Logs Insights 小组件。此外，您在监控账户中创建的告警可用于监视源账户中的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

您可以从控制台或使用 AWS CLI 或 `PutDashboard` API 操作来创建控制面板。您可以将控制面板添加到收藏夹列表。在收藏夹列表中，您不仅可以访问您收藏的控制面板，还可以访问您最近访问过的控制面板。有关更多信息，请参阅将控制面板添加到收藏夹列表。

要访问 CloudWatch 控制面板，您需要以下项目之一：

- `AdministratorAccess` 策略
- `CloudWatchFullAccess` 策略
- 包含以下一个或多个特定权限的自定义策略：
  - `cloudwatch:GetDashboard` 和 `cloudwatch:ListDashboards` 能够查看控制面板
  - `cloudwatch:PutDashboard` 能够创建或修改控制面板
  - `cloudwatch:DeleteDashboards` 能够删除控制面板

内容

- 创建 CloudWatch 控制面板
- CloudWatch 跨账户可观测性控制面板
- 跨账户跨区域的控制面板

- 使用控制面板变量创建灵活的控制面板
- 在 CloudWatch 控制面板上创建和使用小部件
- 共享 CloudWatch 控制面板
- 使用实时数据
- 查看动画控制面板
- 将 CloudWatch 控制面板添加到收藏夹列表
- 更改 CloudWatch 控制面板的时间段覆盖设置或刷新间隔
- 更改 CloudWatch 控制面板的时间范围或时区格式

创建 CloudWatch 控制面板

要开始使用，请创建 CloudWatch 控制面板。您可以创建多个控制面板，而且您可以将它们添加到一个收藏夹列表。您不会受 AWS 账户 内的控制面板数量的限制。所有控制面板都适用于全局。它们不是某个区域所特有的。

以下程序说明如何从 CloudWatch 控制台创建控制面板。您可以在命令行界面中使用 `PutDashboard` API 操作来创建控制面板。API 操作包含对您的控制面板内容进行定义的 JSON 字符串。有关使用 `PutDashboard` API 操作创建控制面板的更多信息，请参阅 Amazon CloudWatch API 参考中的 PutDashboard。

> **Tip**
>
> 如果使用 `PutDashboard` API 操作创建新的控制面板，您可以使用来自现有控制面板的 JSON 字符串。

从控制台创建控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择 Create dashboard（创建控制面板）。
3. 在 Create new dashboard（创建新的控制面板）对话框中，输入控制面板的名称，然后选择 Create dashboard（创建控制面板）。

   如果使用的名称是 CloudWatch-Default 或 CloudWatch-Default-***`ResourceGroupName`***，控制面板将在 CloudWatch 概述主页的 Default Dashboard（默认控制面板）下方显示。有关更多信息，请参阅 Amazon CloudWatch 入门。

4. 在 Add to this dashboard（添加到此控制面板）对话框中执行下列操作之一：
   - 要将图表添加到控制面板，请选择 Line（折线图）或 Stacked area（堆叠面积图），然后选择 Configure（配置）。在 Add metric graph（添加指标图表）对话框中，选择要生成图表的指标，然后选择 Create widget（创建小组件）。如果某指标由于超过 14 天未发布数据而未显示在对话框中，您可以手动添加该指标。有关更多信息，请参阅 在 CloudWatch 控制面板上手动绘制指标图表。
   - 要将显示指标的数字添加到控制面板中，请选择 Number（数字），然后选择 Configure（配置）。在 Add metric graph（添加指标图表）对话框中，选择要生成图表的指标，然后选择 Create widget（创建小组件）。
   - 要将文本块添加到控制面板中，请选择 Text（文本），然后选择 Configure（配置）。在 New text widget（新的文本小组件）对话框中，对于 Markdown，使用 Markdown 设置文本格式，然后选择 Create widget（创建小组件）。
5. （可选）选择 Add widget（添加小组件）并重复步骤 4，将另一个小组件添加到控制面板。您可以多次重复此步骤。

   对于控制面板上的每个图表，右上角都有一个信息图标。选择此图标可查看图表中指标的描述。
6. 选择 Save dashboard。

CloudWatch 跨账户可观测性控制面板

如果您有多个 AWS 账户，则可以设置 CloudWatch 跨账户可观测性，然后在监控账户中创建丰富的跨账户控制面板。您可以无缝搜索、可视化和分析您的指标、日志和跟踪记录，而不受账户限制。

有关设置 CloudWatch 跨账户可观测性的更多信息，请参阅 CloudWatch 跨账户可观测性。

通过 CloudWatch 跨账户可观测性，您可以在监控账户的控制面板中执行以下操作：

- 搜索、查看和创建源账户中的指标图表。一个图表可以包含来自多个账户的指标。
- 在监控账户中创建告警，以监视源账户中的指标。
- 查看位于源账户中的日志组的日志事件，并对源账户中的日志组运行 CloudWatch Logs Insights 查询。监控账户中的单个 CloudWatch Logs Insights 查询可以同时查询多个源账户中的多个日志组。
- 在 X-Ray 中的跟踪地图中查看源账户的节点。然后，您可以将该地图筛选到特定的源账户。

当您登录监控账户时，支持 CloudWatch 跨账户可观测性功能的每个页面的右上角都会出现一个蓝色的 Monitoring account（监控账户）徽章。

跨账户跨区域的控制面板

您可以创建跨账户跨区域的控制面板，这些控制面板可将来自多个 AWS 账户和多个区域的 CloudWatch 数据汇总到一个控制面板中。在此高级控制面板中，您可以了解整个应用程序，并深入了解更多特定的控制面板，而无需登录和注销账户或切换区域。

您可以在 AWS Management Console中和以编程方式创建跨账户跨区域的控制面板。

先决条件

您必须先启用至少一个共享账户和至少一个监控账户，然后才能创建跨账户跨区域的控制面板。此外，要能够使用 CloudWatch 控制台创建跨账户的控制面板，您必须启用控制台以实现跨账户功能。有关更多信息，请参阅 跨账户跨区域的 CloudWatch 控制台。

利用 AWS Management Console 创建和使用跨账户跨区域的控制面板

您可以使用 AWS Management Console创建跨账户跨区域的控制面板。

创建跨账户跨区域的控制面板

1. 登录到监控账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在导航窗格中，选择 Dashboards（控制面板）。
4. 选择一个控制面板，或创建一个新的控制面板。
5. 在屏幕顶部，您可以在账户和区域之间切换。在创建控制面板时，您可以包含来自多个账户和区域的小组件。小组件包括图表、告警和 CloudWatch Logs Insights 小组件。

使用来自不同的账户和区域的指标创建图表

1. 登录到监控账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。
4. 选择要从中添加指标的账户和区域。您可以从屏幕右上角附近的账户和区域下拉菜单中选择您的账户和区域。
5. 将所需指标添加到图表中。有关更多信息，请参阅 绘制指标的图表。
6. 重复步骤 4-5 以添加来自其他账户和区域的指标。

7. （可选）选择 Graphed metrics（绘制的指标）选项卡，然后添加使用了所选指标的指标数学函数。有关更多信息，请参阅 使用指标数学。

   您还可以设置单个图表以包含多个 SEARCH 函数。每项搜索均可引用不同的账户或区域。
8. 在完成图表后，依次选择 Actions (操作) 和 Add to dashboard (添加到控制面板)。

   选择您的跨账户控制面板，然后选择 Add to dashboard (添加到控制面板)。

将来自不同账户的警报添加到跨账户的控制面板

1. 登录到监控账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在页面的顶部，选择告警所在的账户。
4. 在导航窗格中，选择告警。
5. 选中要添加的警报旁边的复选框，然后选择 Add to dashboard (添加到控制面板)。
6. 选择要将警报添加到的跨账户的控制面板，然后选择 Add to dashboard (添加到控制面板)。

以编程方式创建跨账户跨区域的控制面板

您可以使用 AWS API 和 SDK 以编程方式创建控制面板。有关更多信息，请参阅 PutDashboard。

为了启用跨账户跨区域的控制面板，我们已向控制面板正文结构添加新参数，如下表和示例中所示。有关总体控制面板正文结构的更多信息，请参阅控制面板正文结构和语法。

参数	使用	范围	默认
`accountId`	指定小组件或指标所在的账户的 ID。	小组件或指标	当前登录的账户
`region`	指定指标的区域。	小组件或指标	控制台中选定的当前区域

以下示例显示了跨账户跨区域的控制面板中的小组件的 JSON 源。

此示例将 `accountId` 字段设置为小组件级别的共享账户的 ID。这指定了此小组件中的所有指标将来自该共享账户和区域。

```
{
    "widgets": [
        {
            ...
            "properties": {
              "metrics": [
                     …..
                     ],
                    "accountId": "111122223333",
                    "region": "us-east-1"
           }
       }
  ]
}
```

此示例在每个指标级别以不同的方式设置 accountId 字段。在此示例中，此指标数学表达式中的各个指标来自不同的共享账户和不同的区域。

```
{
"widgets": [
    {
        ...
        "properties": {
            "metrics": [
                [ { "expression": "SUM(METRICS())", "label": "[avg: ${AVG}]
 Expression1", "id": "e1", "stat": "Sum" } ],
                [ "AWS/EC2", "CPUUtilization", { "id": "m2", "accountId":
 "5555666677778888", "region": "us-east-1", "label": "[avg: ${AVG}] ApplicationALabel
 " } ],
                [ ".", ".", { "id": "m1", "accountId": "9999000011112222", "region":
 "eu-west-1", "label": "[avg: ${AVG}] ApplicationBLabel" } ]
            ],
            "view": "timeSeries",
            "region":"us-east-1", ---> home region of the metric. Not present in above
 example
            "stacked": false,
            "stat": "Sum",
            "period": 300,
            "title": "Cross account example"
        }
    }
]
```

```
}
```

此示例显示了一个警报小组件。

```
{
    "type": "metric",
    "x": 6,
    "y": 0,
    "width": 6,
    "height": 6,
    "properties": {
        "accountID": "111122223333",
        "title": "over50",
        "annotations": {
            "alarms": [
                "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
            ]
        },
        "view": "timeSeries",
        "stacked": false
    }
}
```

此示例针对的是 CloudWatch Logs Insights 小组件。

```
{
    "type": "log",
    "x": 0,
    "y": 6,
    "width": 24,
    "height": 6,
    "properties": {
        "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp
 desc\n| limit 20",
        "accountId": "111122223333",
        "region": "us-east-1",
        "stacked": false,
        "view": "table"
    }
}
```

以编程方式创建控制面板的另一种方法是，先在 AWS Management Console中创建一个控制面板，然后复制此控制面板的 JSON 源。为此，请加载该控制面板，然后依次选择 Actions (操作) 和 View/edit source (查看/编辑源)。然后，您可以复制此控制面板 JSON 以将其用作模板，从而创建类似的控制面板。

使用控制面板变量创建灵活的控制面板

使用控制面板变量创建灵活的控制面板，从而可以根据控制面板的输入字段中的值，在多个小部件中快速显示不同的内容。例如，您可以创建一个可在不同 Lambda 函数或 Amazon EC2 实例 ID 之间快速切换的控制面板，也可以创建一个可以切换到不同 AWS 区域的控制面板。

创建使用变量的控制面板后，您可以将相同的变量模式复制到其他现有的控制面板。

使用控制面板变量，可以为使用您的控制面板的用户完善操作工作流。此外还可以降低成本，因为是在一个控制面板中使用控制面板变量，而不是创建多个类似的控制面板。

> **Note**
>
> 如果您共享包含控制面板变量的控制面板，则与您共享该控制面板的人将无法更改变量值。

控制面板变量的类型

控制面板变量可以是属性变量，也可以是模式变量。

- 属性变量更改某个属性时，会更改控制面板的所有小部件中出现该属性之处。该属性可以是控制面板的 JSON 源中的任何 JSON 属性，例如 `region`，也可以是指标的维度名称，例如 `InstanceID` 或 `FunctionName`。

  有关使用属性变量的教程，请参阅 教程：将函数名称作为变量创建 Lambda 控制面板。

  有关控制面板的 JSON 源的更多信息，请参阅 控制面板正文结构和语法。在 CloudWatch 控制台中，选择操作、查看/编辑源，即可查看任何自定义控制面板的 JSON 源。
- 模式变量使用正则表达式模式，来更改所有出现某个 JSON 属性之处，或仅更改其中的特定部分。

  有关使用模式变量的教程，请参阅 教程：创建使用正则表达式模式在区域之间切换的控制面板。

属性变量适用于大多数应用场景，设置起来也不太复杂。

主题

- 教程：将函数名称作为变量创建 Lambda 控制面板
- 教程：创建使用正则表达式模式在区域之间切换的控制面板
- 将变量复制到其他控制面板

教程：将函数名称作为变量创建 Lambda 控制面板

此过程中的步骤说明了如何使用属性变量，创建可显示各种指标图表的灵活控制面板。这包括在控制面板上显示下拉选择框，用来在不同的 Lambda 函数之间切换所有图表中的指标。

此类控制面板的其他应用场景示例，包括将 `InstanceId` 作为变量来创建带有实例 ID 下拉列表的指标控制面板。您还可以创建一个控制面板，将 `region` 作为变量来显示来自不同区域的同一组指标。

使用控制面板属性变量创建灵活的 Lambda 控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards、Create dashboard。
3. 输入控制面板的名称，然后选择创建控制面板。
4. 向控制面板添加显示 Lambda 函数指标的小部件。创建这些小部件时，为小部件指标指定 Lambda、按函数名称。对于该函数，请指定要包含在此控制面板中的一个 Lambda 函数。

   有关向控制面板添加小部件的更多信息，请参阅 在 CloudWatch 控制面板上创建和使用小部件。
5. 添加小部件后，在查看控制面板时，选择操作、变量、创建变量。
6. 选择属性变量。
7. 对于变量更改的属性，选择函数名称。
8. 对于输入类型，对于此应用场景，我们建议选择选择菜单（下拉列表）。这将在控制面板中创建一个下拉菜单，让您可以在其中选择要显示指标的 Lambda 函数名称。

   如果此控制面板仅在变量的两三个不同值之间切换，则建议选择单选按钮。

   如果您希望输入或粘贴变量的值，则可以选择文本输入。此选项不包括下拉列表或单选按钮。
9. 选择选择菜单（下拉列表）时，必须选择是通过输入值还是使用指标搜索来填充菜单。对于此应用场景，假设您有大量 Lambda 函数，但不想手动输入所有函数。选择使用指标搜索结果，然后执行以下操作：

   a. 选择预先构建的查询、Lambda、错误。

（选择错误时，不会将错误指标添加到控制面板，而是快速填充函数名称变量选择框。）

b. 选择按函数名称，然后选择搜索。

在搜索按钮下，您将看到已选中函数名称。您还会看到一条关于在输入框中找到多少个函数名称维度值的消息。

10. （可选）要进行更多设置，请选择辅助设置并执行以下一项或多项操作：

- 要自定义变量的名称，请在自定义变量名称中输入名称。
- 要自定义变量输入字段的标签，请在输入标签中输入标签。
- 要设置首次打开控制面板时此变量的默认值，请在默认值中输入默认值。

11. 选择添加变量。

这时将在控制面板顶部附近显示一个函数名称下拉选择框。您可以在此框中选择一个 Lambda 函数，然后所有使用该变量的小部件都将显示有关所选函数的信息。

以后向控制面板添加其他小部件来监视函数名称维度的 Lambda 指标时，这些小部件将自动使用该变量。

教程：创建使用正则表达式模式在区域之间切换的控制面板

此过程中的步骤说明了如何创建可以在区域之间切换的灵活控制面板。本教程使用正则表达式模式变量，而不是属性变量。有关使用属性变量的教程，请参阅 教程：将函数名称作为变量创建 Lambda 控制面板。

对于许多应用场景，您可以创建使用属性变量在区域之间切换的控制面板。但如果小部件依赖包含区域名称的 Amazon 资源名称（ARN），则必须使用模式变量来更改 ARN 中的区域名称。

使用控制面板模式变量创建灵活的多区域控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards、Create dashboard。
3. 输入控制面板的名称，然后选择创建控制面板。
4. 向控制面板添加小部件。添加要显示特定区域数据的小部件时，请不要指定仅在一个区域有值的任何维度。例如，对于 Amazon EC2 指标，应指定聚合指标，而不是 InstanceID 维度的指标。

   有关向控制面板添加小部件的更多信息，请参阅 在 CloudWatch 控制面板上创建和使用小部件。
5. 添加小部件后，在查看控制面板时，选择操作、变量、创建变量。

6. 选择模式变量。
7. 对于变量更改的属性，输入当前控制面板区域的名称，例如 **us-east-2**。

   如果该框下方的标签显示了将该受变量影响的小部件，则说明您输入的区域正确。
8. 对于输入类型，对于此应用场景，选择单选按钮。
9. 对于定义输入的填充方式，选择创建自定义值列表。
10. 在创建自定义值中，输入要切换的区域，每行一个区域。在每个区域之后，输入逗号，然后输入要为该单选按钮显示的标签。例如：

    us-east-1, N. Virginia

    us-east-2, Ohio

    eu-west-3, Paris

    填写自定义值时，预览窗格会更新显示单选按钮的效果。
11. （可选）要进行更多设置，请选择辅助设置并执行以下一项或多项操作：

    - 要自定义变量的名称，请在自定义变量名称中输入名称。
    - 要自定义变量输入字段的标签，请在输入标签中输入标签。在本教程中，请输入 **Region:**。

      在此处输入值时，预览窗格会更新显示单选按钮的效果。
    - 要设置首次打开控制面板时此变量的默认值，请在默认值中输入默认值。
12. 选择添加变量。

    这时会显示控制面板，顶部附近的区域单选按钮旁将显示一个 Region: 标签。在区域之间切换时，所有使用该变量的小部件都将显示有关所选区域的信息。

将变量复制到其他控制面板

使用实用的变量创建控制面板后，您可以将这些变量复制到其他现有的控制面板。有关控制面板变量的更多信息，请参阅 使用控制面板变量创建灵活的控制面板。

将控制面板变量复制到其他控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择控制面板，然后选择包含要复制的变量的控制面板名称。在需要时输入字符串以查找名称相符的控制面板。

3. 选择操作、变量、管理变量。
4. 选择要复制的变量旁的单选按钮，然后选择复制到其他控制面板。
5. 选中该选择框，然后开始键入要将变量复制到其中的控制面板的名称。
6. 选择该控制面板名称，然后选择复制变量。

在 CloudWatch 控制面板上创建和使用小部件

使用本节中的主题，在控制面板中创建和处理图表、告警和文本小组件。

内容

- 在 CloudWatch 控制面板中添加或删除图表
- 在 CloudWatch 控制面板上手动绘制指标图表
- 使用现有图表
- 将 Metrics Explorer 小组件添加到 CloudWatch 控制面板
- 在 CloudWatch 控制面板上添加或删除折线图小组件
- 在 CloudWatch 控制面板上添加或删除数字小组件
- 在 CloudWatch 控制面板上添加或删除量规图小组件
- 将自定义小组件添加到 CloudWatch 控制面板
- 在 CloudWatch 控制面板上添加或删除文本小组件
- 在 CloudWatch 控制面板上添加或删除警报小组件
- 在 CloudWatch 控制面板上添加或删除数据表小组件
- 在 CloudWatch 控制面板上链接和取消链接图表

在 CloudWatch 控制面板中添加或删除图表

您可以将包含一个或多个指标的图表添加到您的 CloudWatch 控制面板。可添加到控制面板的图表类型包括：折线图、堆叠面积图、数字、量规图、条形图和饼图。当您不再需要这些图表时，可以将它们从控制面板中删除。本节内容中的步骤描述如何在您的控制面板中添加和删除图表。如需了解有关如何编辑控制面板上的图表，请参阅在 CloudWatch 控制面板上编辑图表。

将图表添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。

3. 选择 + 符号，然后选择您想要添加到控制面板的图表，然后选择 下一步。

   - 如果您选择 Line（折线图）、Stacked area（堆叠面积图）、Bar（条形图）或 Pie（饼图），请选择 Metrics（指标）。

4. 在浏览选项卡中，搜索或浏览要生成图表的指标，然后选择所需指标。
5. （可选）要更改图表的时间范围，请在屏幕上部选择一个预定义的时间范围。这些时间范围从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。

   要设置自己的时间范围，请选择 Custom（自定义）。

   - （可选）要让此小组件继续使用您选择的时间范围，即使控制面板其余部分的时间范围稍后发生了更改，请选择保留时间范围。

6. （可选）要更改您的图表的小组件类型，请使用预定义时间范围旁边的下拉菜单。
7. （可选）在 Graphed metrics（绘制的指标）中，您可以为指标添加动态标签，然后更改您的指标的标签、标签颜色、统计数据和时间段。您还可以从左到右决定 Y 轴上的标签的位置。

   a. 要添加动态标签，请选择 Graphed metrics（绘制的指标），然后选择 Add dynamic labels（添加动态标签）。动态标签显示关于图表图例中的指标的统计数据。每当刷新控制面板或图表时，动态标签都会自动更新。默认情况下，添加到标签的动态值显示在标签开头。有关更多信息，请参阅 使用动态标签。

   b. 要更改指标的颜色，请选择指标旁边的色块。

   c. 要更改统计数据，请选择 Statistic（统计数据）下方的下拉菜单，然后选择一个新值。有关更多信息，请参阅统计数据。

   d. 要更改时间段，请选择 Period（时间段）列下方的下拉菜单，然后选择一个新值。

8. 如果要创建量规图小组件，则必须选择选项选项卡并指定用于量规图两端的最小值和最大值。
9. （可选）要自定义 Y 轴，请选择 Options（选项）。您可以在标签字段的 Left Y-axis（左 Y 轴）下方添加自定义标签。如果图表在 Y 轴的右侧显示值，您也可以自定义该标签。您还可以在 Y 轴上设置最小值和最大值限制，以便图表仅显示您指定的值范围。
10. （可选）要在折线图或堆叠面积图中添加或编辑横向注释，或者向量规图小组件添加阈值，请选择选项：

    a. 要添加横向注释或阈值，请选择添加横向注释或添加阈值。

    b. 对于标签，输入注释的标签，然后选择对勾图标。

    c. 对于 Value（值），选择当前值旁边的笔和纸图标，然后输入新值。在输入值后，选择对勾图标。

d. 对于 Fill（填充），选择下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。要更改填充颜色，请选择注释旁边的色块。

e. 对于 Axis（轴），指定注释显示在 Y 轴的左侧还是右侧。

f. 要隐藏注释，请清除要隐藏的注释旁边的复选框。

g. 要删除注释，请选择 Actions（操作）下方的 X。

> **Note**
>
> 您可以重复这些步骤以向同一个图表或量规图添加多个横向注释或阈值。

11. （可选）要添加或编辑垂直注释，请选择 Options（选项）：

a. 要添加垂直注释，请选择 Add vertical annotation (添加垂直注释)。

b. 对于 Label（标签），选择当前注释旁边的笔和纸图标，然后输入新的注释。如果您想要仅显示日期和时间，请将标签字段留空。

c. 对于 Date（日期），选择当前的日期和时间，然后输入新的日期和时间。

d. 对于 Fill（填充），选择下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。要更改填充颜色，请选择注释旁边的色块。

e. 要隐藏注释，请清除要隐藏的注释旁边的复选框。

f. 要删除注释，请选择 Actions（操作）下方的 X。

> **Note**
>
> 您可以重复这些步骤以向同一个图表添加多个垂直注释。

12. 选择 Create widget。

13. 选择 Save dashboard。

从控制面板中删除图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。

3. 在要删除的图表的右上角，选择 Widget actions（小组件操作），然后选择 Delete（删除）。
4. 选择 Save dashboard。

在 CloudWatch 控制面板上手动绘制指标图表

如果某个指标在过去 14 天内未发布数据，在搜索要添加到 CloudWatch 控制面板上的图表的指标时，将找不到该指标。执行以下操作步骤手动将指标添加到现有图表：

将在搜索时找不到的指标添加到图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 控制面板必须已包含要添加该指标的图表。如果没有，请创建图表并在其中添加任何指标。有关更多信息，请参阅 在 CloudWatch 控制面板中添加或删除图表 。
4. 依次选择操作、编辑控制面板。

   出现一个 JSON 数据块。该数据块指定控制面板中包含的小部件及其内容。下面的示例是一个定义图表的数据块的一部分。

```
{
            "type": "metric",
            "x": 0,
            "y": 0,
            "width": 6,
            "height": 3,
            "properties": {
                "view": "singleValue",
                "metrics": [
                    [ "AWS/EBS", "VolumeReadOps", "VolumeId",
 "vol-1234567890abcdef0" ]
                ],
                "region": "us-west-1"
            }
        },
```

   此示例中，下面的部分定义了显示在此图表中的指标。

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. 在右方括号后面添加一个逗号（如果没有），然后在逗号后面添加一个带有方括号的类似部分。在该新部分中，指定要添加到图表的指标的命名空间、指标名称以及所需的任何维度。示例如下：

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

   有关指标的 JSON 格式的详细信息，请参阅指标小部件对象的属性。

6. 选择更新。

使用现有图表

按照这些部分中的步骤编辑和修改现有的控制面板图表小组件。

主题

- 在 CloudWatch 控制面板上编辑图表
- 在 CloudWatch 控制面板上移动图表或调整图表的大小
- 在 CloudWatch 控制面板上重命名图表

在 CloudWatch 控制面板上编辑图表

您可以编辑添加到 CloudWatch 控制面板的图表。您可以更改图表的标题、统计数据或时间段。您可以在您的图表中添加、更新和删除指标。如果图表包含不止一个指标，您可以通过隐藏不使用的指标，减少视觉混乱。本节内容中的程序描述如何编辑控制面板上的图表。有关创建图表的信息，请参阅在 CloudWatch 控制面板中添加或删除图表。

New interface

在控制面板上编辑图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在要编辑的图表的右上角，选择 Widget actions（小组件操作），然后选择 Edit（编辑）。
4. 要更改图表的标题，请选择当前标题旁边的笔和纸图标。输入新标题，然后选择 Apply（应用）。

5. （可选）要更改图表的时间范围，请在图表上部选择一个预定义的时间范围。这些时间范围从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。

   要设置自己的时间范围，请选择 Custom（自定义）。

   - （可选）要让此小组件继续使用您选择的时间范围，即使控制面板其余部分的时间范围稍后发生了更改，请选择保留时间范围。

6. 要更改您的图表的小组件类型，请使用预定义时间范围旁边的下拉菜单。

7. 在 Graphed metrics（绘制的指标）中，您可以为指标添加动态标签，然后更改您的指标的标签、标签颜色、统计数据和时间段。您还可以从左到右决定 Y 轴上的标签的位置。

   a. 要为某指标添加动态标签，请选择 Dynamic labels（动态标签）。动态标签显示关于图表图例中的指标的统计数据。每当刷新控制面板或图表时，动态标签都会自动更新。默认情况下，添加到标签的动态值显示在标签开头。有关更多信息，请参阅 使用动态标签。

   b. 要更改指标的颜色，请选择指标旁边的色块。

   c. 要更改统计数据，请选择 Statistic（统计数据）列下的统计值，然后选择一个新值。有关更多信息，请参阅 统计数据。

   d. 要更改时间段，请选择 Period（时间段）列下的时间段值，然后选择一个新值。

8. 要添加或编辑横向注释，请选择 Options（选项）：

   a. 要添加横向注释，请选择 Add horizontal annotation：

   b. 对于 Label（标签），选择当前注释旁边的笔和纸图标。然后输入新注释。在输入注释后，选择对勾图标。

   c. 对于 Value（值），选择当前指标值旁边的笔和纸图标。然后输入新指标值。在输入值后，选择对勾图标。

   d. 对于 Fill（填充），选择列下的下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。如果选择 Between（中间），系统将显示另一个新的标签和值字段。

   > **Tip**
   >
   > 您可以通过选择注释旁边的彩色方块来更改填充颜色。

   e. 对于 Axis（轴），指定注释显示在 Y 轴的左侧还是右侧。

   f. 要隐藏注释，请取消选择要在图表上隐藏的注释旁边的复选框。

g. 要删除注释，请选择 Actions（操作）列下的 X。

> **Note**
>
> 您可以重复这些步骤以向同一个图表添加多个横向注释。

9. 要添加或编辑垂直注释，请选择 Options（选项）：

   a. 要添加垂直注释，请选择 Add vertical annotation (添加垂直注释)。

   b. 对于 Label（标签），选择当前注释旁边的笔和纸图标。然后输入新注释。在输入注释后，选择对勾图标。

   > **Tip**
   >
   > 要仅显示日期和时间，请将标签字段留空。

   c. 对于 Date（日期），选择当前日期和时间。然后，输入新的日期和时间。

   d. 对于 Fill（填充），选择列下的下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。如果选择 Between（中间），系统将显示一个新的标签和值字段。

   > **Tip**
   >
   > 您可以通过选择注释旁边的彩色方块来更改填充颜色。

   > **Note**
   >
   > 您可以重复这些步骤以向同一个图表添加多个垂直注释。

   e. 要隐藏注释，请取消选择要在图表上隐藏的注释旁边的复选框。

   f. 要删除注释，请选择 Actions（操作）列下的 X。

10. 要自定义 Y 轴，请选择 Options（选项）。在 Left Y-axis（左 Y 轴）的下方，您可以为 Label（标签）中输入自定义标签。如果图表在右 Y 轴上显示值，您也可以自定义该标签。您还可以在 Y 轴上设置最小值和最大值，以便图表仅显示您指定的值范围。

11. 完成更改后，选择 Update widget（更新小组件）。

隐藏或更改图表图例的位置

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在要编辑的图表的右上角，选择 Widget actions（小组件操作）。选择 Legend（图例），然后选择 Hidden（隐藏）、Bottom（底部）或 Right（右侧）。

为控制面板上的图表暂时隐藏指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 为您想要在图表脚注中隐藏的指标选择色块。当您将鼠标悬停在色块上时，色块中会显示 X；而当您选中它时，方块会变成灰色。
4. 要恢复隐藏的指标，请清除灰色方块中的 X。

Original interface

在控制面板上编辑图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标悬停在要编辑的图表的右上角。选择 Widget actions（小组件操作），然后选择 Edit（编辑）。
4. 要更改图表的标题，请选择当前标题旁边的铅笔图标，然后输入新的标题。
5. 要更改图表的时间范围，请在图表的上部区域中选择一个预定义的时间范围。这些时间从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。
   - 要设置自己的时间范围，请选择 Custom（自定义）。
6. 要更改图表的小组件类型，请选择 Graph options（图表选项）选项卡。您可以选择 Line（折线图）、Stacked area（堆叠面积图）、Number（数字）、Bar（条形图）或 Pie（饼图）。

> **Tip**
>
> 您可以通过选择预定义时间范围旁边的下拉菜单来更改图表的小组件类型。

7. 在 Graphed metrics（绘制的指标）中，您可以为指标添加动态标签，然后更改您的指标的标签、标签颜色、统计数据和时间段。您还可以从左到右决定 Y 轴上的标签的位置。

   a. 要为某指标添加动态标签，请选择 Dynamic labels（动态标签）。动态标签显示关于图表图例中的指标的统计数据。每当刷新控制面板或图表时，动态标签都会自动更新。默认情况下，添加到标签的动态值显示在标签开头。有关更多信息，请参阅 使用动态标签。

   b. 要更改指标的颜色，请选择指标旁边的色块。

   c. 要更改统计数据，请选择 Statistic（统计数据）列下的统计值，然后选择一个新值。有关更多信息，请参阅 统计数据。

   d. 要更改时间段，请选择 Period（时间段）列下的时间段值，然后选择一个新值。

8. 要添加或编辑横向注释，请选择 Graph options：

   a. 要添加横向注释，请选择 Add horizontal annotation：

   b. 对于 Label（标签），选择当前注释旁边的铅笔图标。然后输入新注释。在输入注释后，选择对勾图标。

   c. 对于 Value（值），选择当前指标值旁边的铅笔图标。然后输入新指标值。在输入值后，选择对勾图标。

   d. 对于 Fill（填充），选择列下的下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。如果选择 Between（中间），系统将显示一个新的标签和值字段。

   > **Tip**
   >
   > 您可以通过选择注释旁边的彩色方块来更改填充颜色。

   e. 对于 Axis（轴），指定注释显示在 Y 轴的左侧还是右侧。

   f. 要隐藏注释，请取消选择要在图表上隐藏的注释旁边的复选框。

   g. 要删除注释，请选择 Actions（操作）列下的 X。

   > **Note**
   >
   > 您可以重复这些步骤以向同一个图表添加多个横向注释。

9. 要添加或编辑垂直注释，请选择 Graph options（图表选项）：

   a. 要添加垂直注释，请选择 Add vertical annotation (添加垂直注释)。

b. 对于 Label（标签），选择当前注释旁边的铅笔图标。然后输入新注释。在输入注释后，选择对勾图标。

> **Tip**
>
> 要仅显示日期和时间，请将标签字段留空。

c. 对于 Date（日期），选择当前日期和时间旁边的铅笔图标。然后，输入新的日期和时间。

d. 对于 Fill（填充），选择列下的下拉菜单，然后指定注释将如何使用阴影。您可以选择 None（无）、Above（上方）、Between（中间）或者 Below（下方）。如果选择 Between（中间），系统将显示一个新的标签和值字段。

> **Tip**
>
> 您可以通过选择注释旁边的彩色方块来更改填充颜色。

> **Note**
>
> 您可以重复这些步骤以向同一个图表添加多个垂直注释。

e. 要隐藏注释，请取消选择要在图表上隐藏的注释旁边的复选框。

f. 要删除注释，请选择 Actions（操作）列下的 X。

10. 要自定义 Y 轴，请选择 Graph options (图表选项)。在 Left Y-axis（左 Y 轴）的下方，您可以为 Label（标签）中输入自定义标签。如果图表在右 Y 轴上显示值，您也可以自定义该标签。您还可以在 Y 轴上设置最小值和最大值，以便图表仅显示您指定的值范围。

11. 完成更改后，选择 Update widget（更新小组件）。

隐藏或更改图表图例的位置

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标悬停在要编辑的图表的右上角，然后选择 Widget actions（小组件操作）。选择 Legend（图例），然后选择 Hidden（隐藏）、Bottom（底部）或 Right（右侧）。

为控制面板上的图表暂时隐藏指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 为您想要在图表脚注中隐藏的指标选择色块。当您将鼠标悬停在色块上时，色块中会显示 X；而当您选中它时，方块会变成灰色。
4. 要恢复隐藏的指标，请清除灰色方块中的 X。

在 CloudWatch 控制面板上移动图表或调整图表的大小

您可以在 CloudWatch 控制面板上安排图表和调整图表的大小。

在控制面板上移动图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 请执行以下操作之一：
   - 将鼠标指针悬停在图表标题的上方，直至选择图标出现。选择图表并将其拖动到控制面板上的新位置。
   - 要将小组件移动到控制面板的左上角或左下角，请选择小组件右上角的垂直省略号，即可打开小组件操作菜单。然后选择移动，然后选择小组件要移动到的位置。
4. 选择 Save dashboard。

调整图表大小

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 要增大或减小尺寸，请将鼠标指针悬停在图表的上方，然后拖动图表的右下角。到达希望的大小时，停止拖放。
4. 选择 Save dashboard。

暂时放大图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。

3. 选择图表。或者，将鼠标指针悬停在图表标题的上方，然后依次选择 Widget actions 和 Enlarge。

在 CloudWatch 控制面板上重命名图表

您可以在控制面板上更改 CloudWatch 分配给图表的默认名称。

在控制面板上重命名图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标指针悬停在图表标题上，选择小部件操作，然后选择编辑。
4. 在 Edit graph 屏幕上靠近顶部的部分，选择图表的标题。
5. 对于标题，输入新的名称，然后选择确定（对勾）。在 Edit graph (编辑图表) 屏幕的右下角，选择 Update widget (更新小部件)。

将 Metrics Explorer 小组件添加到 CloudWatch 控制面板

Metrics Explorer 小组件包括具有相同标签或共享相同资源属性（例如实例类型）的多个资源的图表。创建或删除匹配的资源时，这些小组件会保持最新状态。将 Metrics Explorer 小组件添加到控制面板可帮助您更有效地对环境进行故障排除。

例如，您可以通过分配代表其环境（例如生产或测试）的标签来监控 EC2 实例队列。然后，您可以使用这些标签来筛选和聚合操作指标，例如 `CPUUtilization`，以便了解与每个标签关联的 EC2 实例的运行状况和性能。

以下步骤说明了如何使用控制台将 Metrics Explorer 小组件添加到控制面板。您还可以以编程方式或使用 AWS CloudFormation 将其添加。有关更多信息，请参阅 Metrics Explorer 小组件对象定义和 AWS::CloudWatch::Dashboard。

将 Metrics Explorer 小组件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择要向其添加 Metrics Explorer 小组件的控制面板的名称。
4. 选择 + 符号。
5. 选择 Explorer，然后选择 Next（下一步）。

> **Note**
>
> 您必须选择加入新的控制面板视图才能添加 Metrics Explorer 小组件。要加入，请在导航窗格中选择 Dashboards（控制面板），然后在页面顶部的广告条中选择 try out the new interface（试用新界面）。

6. 请执行以下操作之一：
   - 要使用模板，请选择 Pre-filled Explorer widget（预填充的 Explorer 小组件），然后选择要使用的模板。
   - 要创建自定义可视化，请选择 Empty Explorer widget（空 Explorer 小组件）。
7. 选择创建。

   如果您使用了模板，则该小组件会显示在控制面板上（包含选定的指标）。如果您对 Explorer 小组件和控制面板感到满意，请选择 Save dashboard（保存控制面板）。

   如果您没有使用模板，请继续执行以下步骤。
8. 在 Explorer 下的新小组件的 Metrics（指标）框中，从服务中选择一个指标或所有可用指标。

   选择指标之后，您可以选择性地重复此步骤添加更多指标。
9. 对于选定的每个指标，CloudWatch 会在指标名称后面显示其将随即使用的统计数据。若要更改此统计数据，请选择统计数据名称，然后选择您想要的统计数据。
10. 在 From（从）下，选择标签或资源属性来筛选结果。

    执行此操作后，您可以选择性地重复此步骤以选择更多标签或资源属性。

    如果您选择同一属性的多个值（如两个 EC2 实例类型），则 Explorer 将显示与任一选定属性匹配的所有资源。Explorer 将此选择视为 OR 操作。

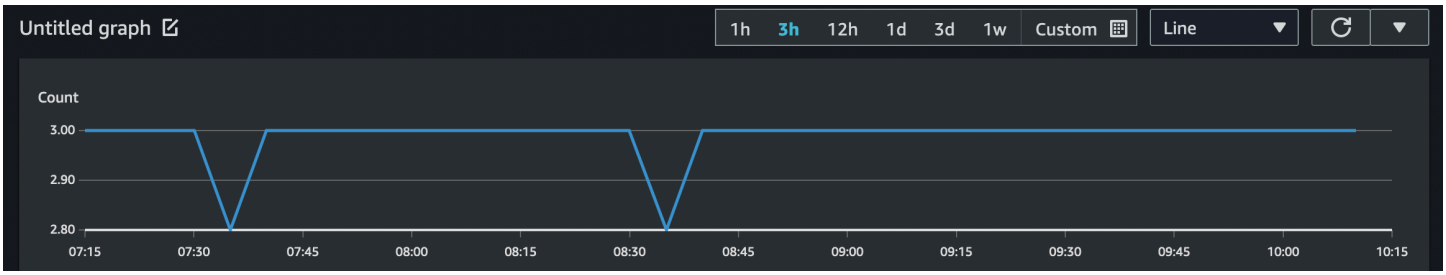
    如果您选择了不同的属性或标签，例如 **`Production`** 标签和 M5 实例类型，则仅显示与所有这些选择匹配的资源。Explorer 将此选择视为 AND 操作。
11. （可选）对于 Aggregate by（聚合依据），选择要用于聚合指标的统计数据。然后，在 for（聚合方式）旁边，选择如何聚合列表中的指标。您可以将当前显示的所有资源聚合在一起，也可以按单个标签或资源属性聚合。

    根据您选择的聚合方式，结果可能是单个时间序列或多个时间序列。
12. 在 Split by（拆分方式）下，您可以选择将具有多个时间序列的单个图表拆分为多个图表。可以按您在 Split by（拆分方式）下选择的多种标准进行拆分。

13. 在 Graph options（绘制选项图表）下，您可以通过更改周期、图表类型、图例放置和布局来优化图表。
14. 如果您对 Explorer 小组件和控制面板感到满意，请选择 Save dashboard（保存控制面板）。

在 CloudWatch 控制面板上添加或删除折线图小组件

使用折线图小组件，您可以比较一段时间内的指标。您还可以使用小组件的缩微贴图缩放功能来检查折线图的各个部分，而无需在放大和缩小视图之间进行切换。本节内容中的程序描述如何在 CloudWatch 控制面板上添加和删除折线图小组件。有关使用小组件的缩微贴图缩放功能和折线图的信息，请参阅放大折线图或堆叠面积图。



将折线图小组件添加到控制面板

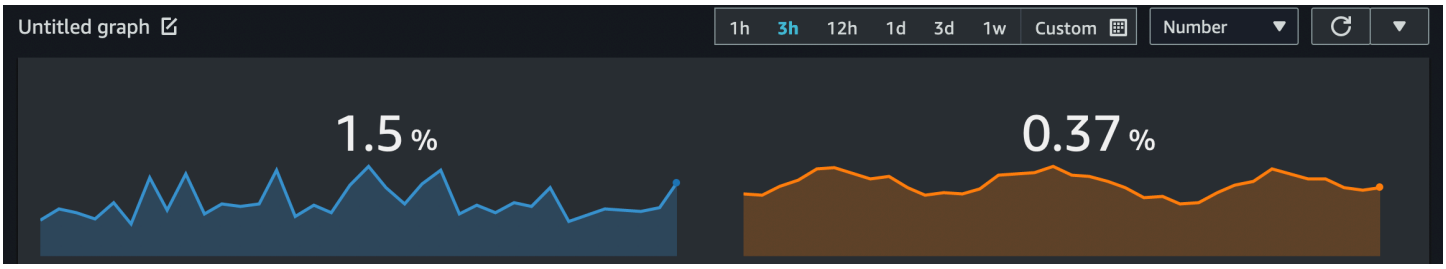
1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择 + 符号，然后选择 Line（线条）。
4. 选择 Metrics（指标）。
5. 选择 Browse（浏览），然后选择您想要生成图表的指标。
6. 选择 Create widget（创建小组件），然后选择 Save dashboard（保存控制面板）。

从控制面板中删除折线图小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在要删除的折线图小组件的右上角，选择 Widget actions（小组件操作），然后选择 Delete（删除）。
4. 选择 Save dashboard。

在 CloudWatch 控制面板上添加或删除数字小组件

使用数字小组件，您可以在最新的指标值和趋势出现后立即查看。由于数字小组件包含走势图功能，因此您可以在单个图表中对指标趋势的上下两部分进行可视化。本节内容中的程序描述如何在 CloudWatch 控制面板中添加和删除数字小组件。



> **Note**
>
> 只有新接口支持走势图功能。当您创建数字小组件时，走势图功能会自动包含在内。

将数字小组件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择 + 符号，然后选择 Number（数字）。
4. 在浏览选项卡中，搜索或浏览要显示的指标。
5. （可选）要更改走势图功能的颜色，请选择 Graphed metrics（绘制的指标），然后选择指标标签旁边的颜色框。将显示一个菜单，供您选择不同的颜色或输入六位十六进制颜色代码，以指定颜色。
6. （可选）要关闭走势图功能，请选择 Options（选项）。在 Sparkline（走势图）的下方，勾选复选框。
7. （可选）要更改数字小组件的时间范围，请在小组件的上部区域中选择一个预定义的时间范围。这些时间范围从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。

   要设置自己的时间范围，请选择 Custom（自定义）。

   - （可选）要让此小组件继续使用您选择的时间范围，即使控制面板其余部分的时间范围稍后发生了更改，请选择保留时间范围。
8. （可选）要让数字小组件显示聚合（1h、3h、12h、1d、3d 或 1w）。

要设置自己的时间范围，请选择 Custom（自定义）。

- （可选）要让此小组件显示整个时间范围内指标值的平均值，而不是最新的值，请选择选项，时间范围值显示整个时间范围的值。

9. 选择 Create widget（创建小组件），然后选择 Save dashboard（保存控制面板）。

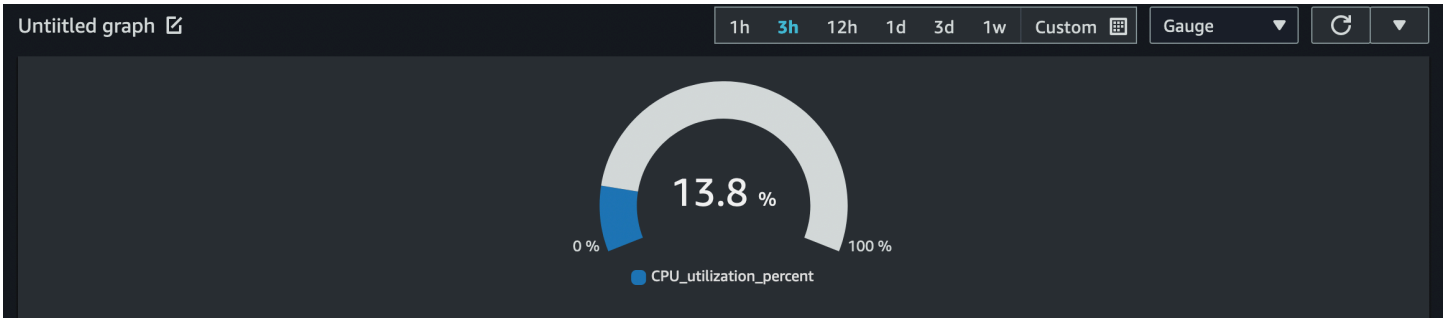
> **Tip**
>
> 您可以在控制面板屏幕的数字小组件中关闭走势图功能。在要修改的数字小组件的右上角，选择 Widget actions（小组件操作）。选择 Sparkline（走势图），然后选择 Hide sparkline（隐藏走势图）。

从控制面板中删除数字小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择包含您想要删除的数字小组件的控制面板。
3. 在要删除的数字小组件的右上角，选择 Widget actions（小组件操作），然后选择 Delete（删除）。
4. 选择 Save dashboard。

在 CloudWatch 控制面板上添加或删除量规图小组件

使用量规图小组件，您可以对介于范围之间的指标值进行可视化。例如，您可以使用量规图小组件来绘制关于百分比和 CPU 使用率的图表，以便在发生任何性能问题时进行观察和诊断。本节内容中的程序描述如何在 CloudWatch 控制面板中添加和删除量规图小组件。



> **Note**
>
> 只有 CloudWatch 控制台中的新接口支持创建量规图小组件。在创建此小组件时，您必须设置一个量规范围。

将量规图小组件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在控制面板屏幕中，选择 + 符号，然后选择 Gauge（量规图）。
4. 选择 Browse（浏览），然后选择您想要生成图表的指标。
5. 选择 Options。在 Gauge range（量规范围）下方，设置 Min（最小）和 Max（最大）值。对于百分比，如 CPU 使用率等，我们建议您将值设置为 `Min` 到 `0` 和 `Max` 到 `100`。
6. （可选）要更改量规图小组件的颜色，请选择 Graphed metrics（绘制的指标），然后选择指标标签旁边的颜色框。将显示一个菜单，供您选择不同的颜色或输入六位十六进制颜色代码，以指定颜色。
7. 选择 Create widget（创建小组件），然后选择 Save dashboard（保存控制面板）。

从控制面板中删除量规图小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择包含您想要删除的量规图小组件的控制面板。
3. 在要删除的量规图小组件的右上角，选择 Widget actions（小组件操作），然后选择 Delete（删除）。
4. 选择 Save dashboard。

将自定义小组件添加到 CloudWatch 控制面板

custom widget（自定义小组件）是一个 CloudWatch 控制面板小组件，它可以使用自定义参数调用任何 AWS Lambda 函数。然后显示返回的 HTML 或 JSON。自定义小组件是一种在控制面板上构建自定义数据视图的简单方法。如果您可以编写 Lambda 代码并创建 HTML，则可以创建有用的自定义小组件。此外，Amazon 提供了几个预构建的自定义小组件，您无需任何代码即可创建这些小组件。

当您创建 Lambda 函数以用作自定义小组件时，我们强烈建议您在函数名称中包含前缀 customWidget。这有助于您在将自定义小组件添加到控制面板时，了解哪些 Lambda 函数可以安全使用。

自定义小组件的操作类似于控制面板上的其他小组件。它们可以刷新和自动刷新、调整大小和移动。它们会对控制面板的时间范围做出反应。

如果您已设置 CloudWatch 控制台跨账户功能，则可以将在一个账户中创建的自定义小组件添加到其他账户中的控制面板。有关更多信息，请参阅 跨账户跨区域的 CloudWatch 控制台。

您还可以使用 CloudWatch 控制面板共享功能在自己的网站上使用自定义小组件。有关更多信息，请参阅 共享 CloudWatch 控制面板。

主题

- 有关自定义小组件的详细信息
- 安全与 JavaScript
- 自定义小组件中的交互性
- 创建自定义小组件
- 示例自定义小组件

有关自定义小组件的详细信息

自定义小组件的工作方式如下：

1. CloudWatch 控制面板调用包含小组件代码的 Lambda 函数。并传递在小组件中定义的任何自定义参数。
2. Lambda 函数返回 HTML、JSON 或 Markdown 字符串。Markdown 以 JSON 格式返回，格式如下：

   ```
   {"markdown":"markdown content"}
   ```

3. 控制面板显示返回的 HTML 或 JSON。

如果函数返回 HTML，则支持大多数 HTML 标签。您可以使用 Cascading 样式表 (CSS) 样式和可扩展矢量图形 (SVG) 来构建复杂的视图。

HTML 元素（例如链接和表格）的默认样式遵循 CloudWatch 控制面板的样式。您可以通过使用内联样式和 `<style>` 标签来自定义此样式。您还可以通过包含具有 `cwdb-no-default-styles` 类的

单个 HTML 元素，停用默认样式。以下示例会停用默认样式： 。

自定义小组件对 Lambda 的每次调用都包含一个具有以下内容的 widgetContext 元素，以便为 Lambda 函数开发人员提供有用的上下文信息。

```
{
    "widgetContext": {
        "dashboardName": "Name-of-current-dashboard",
        "widgetId": "widget-16",
        "accountId": "012345678901",
        "locale": "en",
        "timezone": {
            "label": "UTC",
            "offsetISO": "+00:00",
            "offsetInMinutes": 0
        },
        "period": 300,
        "isAutoPeriod": true,
        "timeRange": {
            "mode": "relative",
            "start": 1627236199729,
            "end": 1627322599729,
            "relativeStart": 86400012,
            "zoom": {
                "start": 1627276030434,
                "end": 1627282956521
             }
        },
        "theme": "light",
        "linkCharts": true,
        "title": "Tweets for Amazon website problem",
        "forms": {
            "all": {}
        },
        "params": {
            "original": "param-to-widget"
        },
        "width": 588,
        "height": 369
    }
}
```

默认 CSS 样式

自定义小组件提供以下默认 CSS 样式元素：

- 您可以使用 CSS 类 btn 添加按钮。它将锚点 (<a>) 转换为按钮，如以下示例所示：

```
<a class="btn" href=https://amazon.com”>Open Amazon</a>
```

- 您可以使用 CSS 类 btn btn-primary 添加主按钮。
- 默认情况下，以下元素采用样式：table（表）、select（选择）、headers (h1, h2, and h3)（标头（标头 1、标头 2 和 标头 3））、preformatted text (pre)（预先格式化的文本（预先））、input（输入）和 text area（文本区域）。

使用描述参数

我们强烈建议您在函数中支持 describe（描述）参数，即使它只返回一个空字符串。如果您不支持它，并在自定义小组件中将其调用，它会像显示文档一样显示小组件内容。

如果您包含 describe（描述）参数，Lambda 函数以 Markdown 格式返回文档，而不执行任何其他操作。

当您在控制台中创建自定义小组件时，在选择 Lambda 函数后，将出现一个 Get documentation（获取文档）按钮。如果选择此按钮，则会使用 describe（描述）参数调用该函数并返回该函数的文档。如果文档在 Markdown 中格式正确，CloudWatch 会解析文档中由 YAML 中的三个反引号字符 (```) 引起的第一个条目。然后，它会自动填充参数中的文档。以下是此类格式正确的文档示例。

```
``` yaml
echo: <h1>Hello world</h1>
```
```

安全与 JavaScript

出于安全原因，返回的 HTML 中不允许使用 JavaScript。删除 JavaScript 可防止出现权限升级问题，即 Lambda 函数的编写者注入的代码可能以比在控制面板上查看小组件的用户更高的权限运行。

如果返回的 HTML 包含任何 JavaScript 代码或其他已知的安全漏洞，则会在将其呈现在控制面板上之前，将其从 HTML 中清除。例如，<iframe> 和 <use> 标签是不允许的，将被删除。

默认情况下，自定义小组件不会在控制面板中运行。相反，如果您信任自定义小组件调用的 Lambda 函数，则必须明确允许其运行。对于单个小组件和整个控制面板，您可以选择允许一次或始终允许。您还可以拒绝对单个小组件和整个控制面板的权限。

自定义小组件中的交互性

尽管不允许使用 JavaScript，但还有其他方法允许与返回的 HTML 进行交互。

- 返回的 HTML 中的任何元素都可以在 <cwdb-action> 标签中使用特殊配置进行标记，该配置可以在弹出窗口中显示信息，在点击时要求确认，并在选择该元素时调用任何 Lambda 函数。例如，您可以定义使用 Lambda 函数调用任何 AWS API 的按钮。可以将返回的 HTML 设置为替换现有 Lambda 小组件的内容，或在模式中显示。
- 返回的 HTML 可以包含打开新控制台、打开其他客户页面或加载其他控制面板的链接。
- HTML 可以包含元素的 `title` 属性，如果用户将鼠标悬停在该元素上，它会显示更多信息。
- 元素可以包含 CSS 选择器，如 `:hover`，它可以调用动画或其他 CSS 效果。该元素可以包含 CSS 选择器，例如，可以调用动画或其他 CSS 效果。您还可以在页面中显示或隐藏元素。您还可以显示或隐藏页面中的元素。

<cwdb-action> 定义和用法

<cwdb-action> 元素定义了前一个元素的行为。<cwdb-action> 的内容为要显示的 HTML 或要传递给 Lambda 函数的参数的 JSON 数据块。

以下是 <cwdb-action> 元素的示例。

```
<cwdb-action
     action="call|html"
     confirmation="message"
     display="popup|widget"
     endpoint="<lambda ARN>"
     event="click|dblclick|mouseenter">

     html | params in JSON
</cwdb-action>
```

- 操作 – 有效值为 `call`，会调用 Lambda 函数并且显示包含在 <cwdb-action> 中的任何 HTML 的 `html`。默认为 `html`。
- 确认 – 显示在执行操作之前必须确认的确认消息，允许客户取消。

- 显示 – 有效值为 popup 和 widget，会替换小组件本身的内容。默认为 widget。
- 端点 – 要调用的 Lambda 函数的 Amazon Resource Name (ARN)。如果 action 是 call，则这是必需的。
- 事件 – 定义上一个调用操作的元素上的事件。有效值包括 click、dblclick 和 mouseenter。mouseenter 事件仅能与 html 操作结合使用。默认为 click。

示例

以下示例说明了如何使用 <cwdb-action> 标签创建按钮，该按钮可使用 Lambda 函数调用重启 Amazon EC2 实例。此选项会在弹出窗口中显示呼叫成功或失败。

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
       { "instanceId": "i-342389adbfef" }
</cwdb-action>
```

下一个示例会在弹出窗口中显示更多信息。

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
   <h1>Big title</h1>
   More info about <b>something important</b>.
</cwdb-action>
```

此示例为 Next（下一步）按钮，该按钮使用对 Lambda 函数的调用替换小组件的内容。

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:nextPage">
   { "pageNum": 2 }
</cwdb-action>
```

创建自定义小组件

要创建自定义小部件，您可以使用 AWS 提供的任意一种示例，也可以创建自己的小部件。AWS 示例包括使用 JavaScript 和 Python 的示例，并且这些示例是通过 AWS CloudFormation 堆栈创建的。有关示例列表，请参阅 示例自定义小组件。

在 CloudWatch 控制面板中创建自定义小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择 + 符号。
4. 选择 Custom widget（自定义小组件）。
5. 使用以下方法之一：
   - 要使用 AWS 提供的自定义小部件，请执行以下操作：
     a. 在下拉框中选择示例。

        AWS CloudFormation 控制台在新浏览器中启动。在 AWS CloudFormation 控制台中，执行以下操作：

     b. （可选）自定义 AWS CloudFormation 堆栈名称。
     c. 选择所示例使用的任何参数。
     d. 选择 I acknowledge that AWS CloudFormation might create IAM resources（我确认，Amazon CloudFormation 可能创建 IAM 资源），然后选择 Create stack（创建堆栈）。
   - 要自行创建 AWS 提供的自定义小组件，请执行以下操作：
     a. 选择下一步。
     b. 选择从列表中选择 Lambda 函数，或输入其 Amazon Resource Name (ARN)。如果从列表中选择它，还要指定函数所在的区域以及要使用的版本。
     c. 对于 Parameters（参数），选择函数所使用的任何参数。
     d. 输入小组件的标题。
     e. 对于 Update on（更新），配置应该更新小组件的时间（应该再次调用 Lambda 函数的时间）。此选项可以是以下一个或多个项：Refresh（刷新）以便在控制面板自动刷新时更新它，Resize（调整大小）以便在调整小组件大小时更新它，或者 Time Range（时间范围）以便在调整控制面板的时间范围时更新它（包括缩放图表时）。
     f. 如果对预览满意，请选择 Create widget（创建小组件）。

示例自定义小组件

AWS 在 JavaScript 和 Python 中提供了示例自定义小组件。您可以使用此列表中各个小组件的链接来创建这些示例小组件。您也可以使用 CloudWatch 控制台创建和自定义小组件。此列表中的链接会打开 AWS CloudFormation 控制台并使用 AWS CloudFormation 快速创建链接来创建自定义小组件。

您还可以在 GitHub 上访问自定义小组件示例。

在此列表之后，显示了每种语言的 Echo 小组件的完整示例。

JavaScript

JavaScript 中的示例自定义小组件

- Echo – 一个基本的回显器，可用于测试 HTML 在自定义小组件中的显示方式，而无需编写新的小组件。
- Hello world – 一个非常基本的入门小组件。
- 自定义小组件调试程序 – 一个调试程序小组件，会显示有关 Lambda 运行时环境的有用信息。
- 查询 CloudWatch Logs Insights – 运行和编辑 CloudWatch Logs Insights 查询。
- 运行 Amazon Athena 查询 – 运行和编辑 Athena 查询。
- 调用 AWS API – 调用任何只读 AWS API 并以 JSON 格式显示结果。
- 快速 CloudWatch 位图图表 – 在服务器端使用渲染 CloudWatch 图表，以便快速显示。
- CloudWatch 控制面板中的文本小组件 – 显示指定 CloudWatch 控制面板中的第一个文本小组件。
- CloudWatch 指标数据作为表 – 在表中显示原始 CloudWatch 指标数据。
- Amazon EC2 表 – 按 CPU 使用率显示排名靠前的 EC2 实例。此小组件还包括一个重启按钮，该按钮默认处于禁用状态。
- AWS CodeDeploy 部署 – 显示 CodeDeploy 部署。
- AWS Cost Explorer 报告 – 显示选定时间范围内每项 AWS 服务的成本报告。
- 显示外部 URL 的内容 – 显示可从外部访问的 URL 的内容。
- 显示 Amazon S3 对象 – 显示您账户中 Amazon S3 存储桶中的对象。
- 简单的 SVG 饼图 – 基于 SVG 的图表小组件示例。

Python

Python 中的示例自定义小组件

- Echo – 一个基本的回显器，可用于测试 HTML 在自定义小组件中的显示方式，而无需编写新的小组件。
- Hello world – 一个非常基本的入门小组件。
- 自定义小组件调试程序 – 一个调试程序小组件，会显示有关 Lambda 运行时环境的有用信息。
- 调用 AWS API – 调用任何只读 AWS API 并以 JSON 格式显示结果。
- 快速 CloudWatch 位图图表 – 在服务器端使用渲染 CloudWatch 图表，以便快速显示。
- 通过电子邮件发送控制面板快照 – 拍摄当前控制面板的快照，并将其发送给电子邮件收件人。
- 向 Amazon S3 发送控制面板快照 – 拍摄当前控制面板的快照，并将其存储在 Amazon S3 中。
- CloudWatch 控制面板中的文本小组件 – 显示指定 CloudWatch 控制面板中的第一个文本小组件。
- 显示外部 URL 的内容 – 显示可从外部访问的 URL 的内容。
- RSS 读取器 – 显示 RSS 馈送。
- 显示 Amazon S3 对象 – 显示您账户中 Amazon S3 存储桶中的对象。
- 简单的 SVG 饼图 – 基于 SVG 的图表小组件示例。

JavaScript 中的 Echo 小组件

以下是 JavaScript 中的 Echo 示例小组件。

```
const DOCS = `
## Echo
A basic echo script. Anything passed in the \`\`\`echo\`\`\` parameter is returned as
 the content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
\`\`\` yaml
echo: <h1>Hello world</h1>
\`\`\`
```

```
`;

exports.handler = async (event) => {
    if (event.describe) {
        return DOCS;
    }

    let widgetContext = JSON.stringify(event.widgetContext, null, 4);
    widgetContext = widgetContext.replace(/</g, '&lt;');
    widgetContext = widgetContext.replace(/>/g, '&gt;');

    return `${event.echo || ''}<pre>${widgetContext}</pre>`;
};
```

Python 中的 Echo 小组件

以下是 Python 中的 Echo 示例小组件。

```
import json

DOCS = """
## Echo
A basic echo script. Anything passed in the ```echo``` parameter is returned as the
 content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```"""

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '<')
    widgetContext = widgetContext.replace('>', '>')
```

```
    return f'{echo}<pre>{widgetContext}</pre>'
```

Java 中的 Echo 小组件

以下是 Java 中的 Echo 示例小组件。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

  static String DOCS = ""
    + "## Echo\n"
    + "A basic echo script. Anything passed in the ```echo``` parameter is returned as
 the content of the custom widget.\n"
    + "### Widget parameters\n"
    + "Param | Description\n"
    + "---|---\n"
    + "**echo** | The content to echo back\n\n"
    + "### Example parameters\n"
    + "```yaml\n"
    + "echo: <h1>Hello world</h1>\n"
    + "```\n";

  Gson gson = new GsonBuilder().setPrettyPrinting().create();

  @Override
  public String handleRequest(Event event, Context context) {

    if (event.describe) {
      return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
 gson.toJson(event.widgetContext) + "</pre>";
  }
}

class Event {
```

```
    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}
```

在 CloudWatch 控制面板上添加或删除文本小组件

文本小部件包含 Markdown 格式的文本块。您可以在 CloudWatch 控制面板中添加、编辑或删除文本小组件。

将文本小部件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择 + 符号。
4. 选择文本。
5. 对于 Markdown，使用 Markdown 添加文本并设置格式，然后选择 Create widget (创建小部件)。
6. 要使文本小组件透明，请选择透明背景。
7. 选择 Save dashboard。

在控制面板上编辑文本小部件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标指针悬停在文本块的右上角，然后选择 Widget actions（小组件操作）。然后选择 Edit（编辑）。
4. 根据需要更新文本，然后选择 Update widget (更新小部件)。
5. 选择 Save dashboard。

从控制面板中删除文本小部件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标指针悬停在文本块的右上角，然后选择 Widget actions（小组件操作）。然后选择 Delete(删除)。
4. 选择 Save dashboard。

在 CloudWatch 控制面板上添加或删除警报小组件

要将警报小组件添加到控制面板，请选择下列选项之一：

- 在小组件中添加单个警报，该小组件会显示警报指标的图表和警报状态。
- 添加一个警报状态小组件，它可以在一个网格中显示多个警报的状态。仅显示警报名称和当前状态，不显示图表。一个警报状态小组件最多可包含 100 个警报。

将单个告警（包括其图表）添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Alarms，选择要添加的警报，然后选择 Add to Dashboard。
3. 选择控制面板，选择小部件类型 (Line、Stacked area 或 Number)，然后选择 Add to dashboard。
4. 要在控制面板上查看警报，请在导航窗格中选择 Dashboards，然后选择控制面板。
5. (可选) 要暂时放大警报图表，请选择该图表。
6. （可选）要更改小组件类型，请将鼠标指针悬停在图表标题上，选择小组件操作，然后选择小组件类型。

将告警状态小组件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择 + 符号。
4. 选择 Alarm status（告警状态）。
5. 选择要向小组件添加的告警旁边的复选框，然后选择 Create widget（创建小组件）。
6. 选择 Add to dashboard（添加到控制面板）。

从控制面板中删除告警小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 将鼠标悬停在小组件上，选择小组件操作，然后选择删除。
4. 选择 Save dashboard。如果在保存更改之前尝试离开控制面板，将提示您保存或放弃更改。

在 CloudWatch 控制面板上添加或删除数据表小组件

您可以通过数据表小组件查看指标的原始数据点以及该原始数据的快速摘要。数据表小组件的图表完全基于实际数据，因此更容易理解所呈现的数据点。本节内容中的程序描述如何在 CloudWatch 控制面板中添加和删除数据表小组件。下图显示一个表小组件的示例，其中包含一组 CloudWatch 指标的“最小值”、“最大值”、“总和”和“平均值”统计数据列。

Label	Min	Max	Sum	Average	11/20 06:00	11/20 00:00	11/19 18:00	11/19 12:00	11/ 06:
TestMetric295	991	1,000	12k	998	996	1,000	997	999	
TestMetric296	995	1,000	12k	998	995	1,000	1,000	998	
TestMetric297	991	1,000	12k	998	998	1,000	999	997	
TestMetric298	994	1,000	12k	997	996	999	995	995	
TestMetric3	993	1,000	12k	998	1,000	999	999	1,000	
TestMetric299	995	999	12k	998	999	995	999	998	
TestMetric30	994	999	12k	998	999	998	999	999	
StackMetric2	99	99.9	1.2k	99.6	99.2	99.7	99.5	99.8	
StackMetric20	99	100	1.19k	99.5	100	99.1	99.4	99.4	
StackMetric21	97.5	100	1.19k	99.4	99.6	99.7	97.6	99.8	

表属性

数据表有一组默认的属性，这些属性无需对选项或源进行任何更改。这些属性包括粘性标签列、所有启用的摘要列、四舍五入的数据点，以及其转换的单位。

每个数据表小组件都可以具有以下属性。有关每个属性的信息包括了如何在控制面板的 JSON 源中对该属性进行配置。有关控制面板 JSON 的更多信息，请参阅 Dashboard Body Structure and Syntax。

摘要

摘要列是数据表小组件中引入的新属性。这些列是当前表格摘要的特定子集。例如，总计摘要是其行中显示的所有数据点的总和。摘要列与 CloudWatch 统计数据不同。源代码表示为：

```
"table": {
```

```
        "summaryColumns": [
            "MIN",
            "MAX",
            "SUM",
            "AVG"
        ]
    },
```

阈值

可通过其将阈值应用到表格中。当数据点处于阈值范围之内，其单元格将以阈值颜色突出显示。源代码表示为：

```
"annotations": {
    "horizontal": [
        {
            "label": string,
            "value": int,
            "fill": "above" | "below"
        }
    ]
}
```

标签列中的单位

要显示与指标关联的单位，您可以启用此选项以在标签旁边的标签列中显示相应单位。源代码表示为：

```
"yAxis": {
    "left": {
        "showUnits": true | false
    }
}
```

反转行和列

这将转换表格，使数据点从列变为行，而指标则从行变为列。源代码表示为：

```
 "table": {
    "layout": "vertical" | "horizontal"
}
```

粘性摘要列

这将使摘要列具有粘性，以便在其滚动时仍能显示。标签已经具有粘性。源代码表示为：

```
"table": {
    "stickySummary": true | false
}
```

仅显示摘要列

这将使数据点列不显示，而是只显示标签列和摘要列。源代码表示为：

```
 "table": {
    "showTimeSeriesData": false | true
}
```

实时数据

显示最新的数据点，即使其尚未完全聚合。源代码表示为：

```
"liveData": true | false
```

数字小组件格式

在四舍五入和转换之前，尽可能多地显示单元格中可容纳的数字。源代码表示为：

```
"singleValueFullPrecision": true | false
```

将数据表小组件添加到控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择控制面板，然后选择一个控制面板。
3. 选择 + 按钮，选择数据表，然后选择下一步。
4. 在浏览选项卡中，搜索或浏览要在表格小组件中显示的指标。然后选择指标。
5. （可选）要更改表的布局，请选择选项选项卡，然后选择反转行和列。

   您也可以通过选项选项卡更改表中显示的列，并显示标签列中使用的单位。

> **Tip**
>
> 要显示更准确的阈值，请选择 Show as many digits as can fit before rounding。

6. （可选）要更改数据表小组件的时间范围，请在小组件的上部区域中选择一个预定义的时间范围。时间范围从 1 小时到 1 周不等。要设置自己的时间范围，请选择 Custom（自定义）。
7. （可选）要更改数据表小组件的时间范围，请在小组件的上部区域中选择一个预定义的时间范围。时间范围从 1 小时到 1 周不等。要设置自己的时间范围，请选择 Custom（自定义）。
8. （可选）要让此小组件继续使用您选择的该时间范围（即使控制面板其余部分的时间范围稍后会发生更改时也是如此），请选择保持时间范围。
9. 选择创建小组件，然后选择保存控制面板。

将表小组件从控制面板中删除

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在要删除的小组件的右上角，选择小组件操作，然后选择删除。
4. 选择 Save dashboard。

在 CloudWatch 控制面板上链接和取消链接图表

可以在控制面板上将图表链接在一起，这样当您放大或缩小一个图表时，其他图表将同时放大或缩小。可以取消链接图表以仅允许缩放一个图表。

在控制面板上链接图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择操作，然后选择链接图表。

在控制面板上取消链接图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。

3. 清除操作，然后清除链接图表。

共享 CloudWatch 控制面板

您可以与无法直接访问您 AWS 账户的人员共享 CloudWatch 控制面板。这使您能够跨团队、与利益攸关方以及企业外部人员共享控制面板。您甚至可以在团队区域的大屏幕上显示控制面板，或将控制面板嵌入到 Wiki 和其他网页中。

> **Warning**
>
> 向与您共享控制面板的所有人员授予该账户中 授予与您共享控制面板的人员的权限 列出的权限。如果您公开共享控制面板，则拥有控制面板链接的所有人均具有这些权限。`cloudwatch:GetMetricData` 和 `ec2:DescribeTags` 权限范围不能缩小到特定指标或 EC2 实例，因此具有控制面板访问权限的人员可以查询所有 CloudWatch 指标以及账户中所有 EC2 实例的名称和标签。

共享控制面板时，您可以通过三种方式指定谁可以查看控制面板：

- 共享单个控制面板并指定最多五个可查看控制面板的人员的电子邮件地址。这些用户中的每个用户都会创建自己的密码，他们必须输入该密码才能查看控制面板。
- 公开共享单个控制面板，以便拥有该链接的任何人都可以查看控制面板。
- 共享您账户中的所有 CloudWatch 控制面板，并指定第三方单点登录 (SSO) 提供商进行控制面板访问。属于此 SSO 提供商列表成员的所有用户均可访问该账户中的所有控制面板。要启用此功能，请将 SSO 提供商与 Amazon Cognito 集成。SSO 提供商必须支持安全断言标记语言 (SAML)。有关 Amazon Cognito 的更多信息，请参阅什么是 Amazon Cognito？

共享控制面板不会产生费用，但共享控制面板内的小组件将按标准的 CloudWatch 费率收费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

当您共享控制面板时，系统会在美国东部（弗吉尼亚州北部）区域创建 Amazon Cognito 资源。

> **Important**
>
> 请勿修改控制面板共享过程创建的资源名称和标识符。这包括 Amazon Cognito 和 IAM 资源。修改这些资源可能会导致共享控制面板出现意外和不正确的功能。

> **Note**
>
> 如果您共享的控制面板中有带警报注释的指标小组件，共享该控制面板的人员将不会看到这些小组件。相反，他们会看到一个空白的小组件，以及说明该小组件不可用的文字。当您自行查看控制面板时，依然可以看到带有警报注释的指标小组件。

共享控制面板所需的权限

为了能够使用以下任一方法来共享控制面板并查看已共享哪些控制面板，您必须以用户身份或使用具有特定权限的 IAM 角色进行登录。

为了能够共享控制面板，您的用户或 IAM 角色必须包含以下策略语句中包含的权限：

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy",
        "iam:PassRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CWDBSharing*",
        "arn:aws:iam::*:policy/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp:*",
        "cognito-identity:*",
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetDashboard",
    ],
```

```
    "Resource": [
        "*"
        // or the ARNs of dashboards that you want to share
    ]
}
```

为了能够查看已共享的控制面板但无法共享控制面板，用户或 IAM 角色可以包含类似于以下内容的策略语句：

```
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp:*",
        "cognito-identity:*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:ListDashboards",
    ],
    "Resource": [
        "*"
    ]
}
```

授予与您共享控制面板的人员的权限

当您共享控制面板时，CloudWatch 会在账户中创建一个 IAM 角色，该角色会向与您共享控制面板的人员授予以下权限：

- `cloudwatch:GetInsightRuleReport`
- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

> **Warning**
>
> 向与您共享控制面板的所有人员授予该账户的这些权限。如果您公开共享控制面板，则拥有控制面板链接的所有人均具有这些权限。
> `cloudwatch:GetMetricData` 和 `ec2:DescribeTags` 权限范围不能缩小到特定指标或 EC2 实例，因此具有控制面板访问权限的人员可以查询所有 CloudWatch 指标以及账户中所有 EC2 实例的名称和标签。

共享控制面板时，默认情况下，CloudWatch 创建的权限仅限制对共享控制面板上的告警和 Contributor Insights 规则的访问。如果您向控制面板添加新告警或 Contributor Insights 规则，并希望与您共享控制面板的人员也能查看，则必须更新策略以允许这些资源。

与特定用户共享单个控制面板

使用本节中的步骤，以便与您选择的最多五个电子邮件地址共享控制面板。

> **Note**
>
> 默认情况下，控制面板上的任何 CloudWatch Logs 小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看日志表小组件。
> 默认情况下，控制面板上的任何复合告警小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看复合告警。

与特定用户共享控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 在 Share your dashboard and require a username and password（共享您的控制面板且需要用户名和密码）旁边，选择 Start sharing（开始共享）。
6. 在 Add email addresses（添加电子邮件地址）下，输入要与其共享控制面板的电子邮件地址。您可以包含最多五个电子邮件地址。
7. 输入所有电子邮件地址后，阅读协议并选择确认框。然后，选择 Preview policy（预览策略）。

8. 确认将要共享的资源是您想要的，然后选择 Confirm and generate shareable link（确认并生成可共享的链接）。
9. 在下一页上，选择 Copy link to clipboard（将链接复制到剪贴板）。然后，您可以将此链接粘贴到电子邮件中并将其发送给受邀用户。他们会自动收到一封带有用户名和临时密码的单独电子邮件，用于连接到控制面板。

公开共享单个控制面板

按照本节中的步骤公开共享控制面板。这十分有利于在团队会议室的大屏幕上显示控制面板或嵌入 Wiki 页面。

> **Important**
>
> 公开共享控制面板使得拥有该链接的任何人都可以访问该控制面板，而无需身份验证。仅对不包含敏感信息的控制面板执行此操作。

> **Note**
>
> 默认情况下，控制面板上的任何 CloudWatch Logs 小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看日志表小组件。
> 默认情况下，控制面板上的任何复合告警小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看复合告警。

公开共享控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 在 Share your dashboard publicly（公开共享您的控制面板）旁，选择 Start sharing（开始共享）。
6. 在文本框中输入 **`Confirm`**。
7. 阅读协议并选择确认框。然后，选择 Preview policy（预览策略）。

8. 确认将要共享的资源是您想要的，然后选择 Confirm and generate shareable link（确认并生成可共享的链接）。
9. 在下一页上，选择 Copy link to clipboard（将链接复制到剪贴板）。然后，您可以共享此链接。与您共享链接的任何人都可以访问控制面板，而无需提供凭证。

使用 SSO 共享账户中的所有 CloudWatch 控制面板

使用本节中的步骤，通过单点登录 (SSO) 与用户共享您账户中的所有控制面板。

> **Note**
>
> 默认情况下，控制面板上的任何 CloudWatch Logs 小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看日志表小组件。
> 默认情况下，控制面板上的任何复合告警小组件对与您共享控制面板的人员均不可见。有关更多信息，请参阅 允许与您共享的人员查看复合告警。

与 SSO 提供商列表中的用户共享您的 CloudWatch 控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 选择 Go to CloudWatch Settings（转到 CloudWatch 设置）。
6. 如果您需要的 SSO 提供商未在 Available SSO providers（可用 SSO 提供商）中列出，请选择 Manage SSO providers（管理 SSO 提供商）并按照 为 CloudWatch 控制面板共享设置 SSO 中的说明进行操作。

   然后返回到 CloudWatch 控制台并刷新浏览器。您启用的 SSO 提供商现在应显示在列表中。
7. 在 Available SSO providers（可用 SSO 提供商）列表中选择所需的 SSO 提供商。
8. 选择 Save changes（保存更改）。

为 CloudWatch 控制面板共享设置 SSO

若要通过支持 SAML 的第三方单点登录提供商设置控制面板共享，请按照下列步骤操作。

> **Important**
>
> 我们强烈建议您不要使用非 SAML SSO 提供商共享控制面板。这样操作会导致无意中允许第三方访问您账户的控制面板的风险。

设置 SSO 提供商以启用控制面板共享

1. 将 SSO 提供商与 Amazon Cognito 集成。有关更多信息，请参阅将第三方 SAML 身份提供商与 Amazon Cognito 用户池集成。
2. 从 SSO 提供商下载元数据 XML 文件。
3. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
4. 在导航窗格中，选择 Settings（设置）。
5. 在 Dashboard sharing（控制面板共享）部分中，选择 Configure（配置）。
6. 选择 Manage SSO providers（管理 SSO 提供商）。

   此操作会在美国东部（弗吉尼亚北部）区域 (us-east-1) 中打开 Amazon Cognito 控制台。如果您没有看到任何 User Pools（用户池），则 Amazon Cognito 控制台可能已在其他区域中打开。如果是，请将区域更改为 US East (N. Virginia) us-east-1（美国东部（弗吉尼亚北部）us-east-1），然后继续执行后续步骤。
7. 选择 CloudWatchDashboardSharing 池。
8. 在导航窗格中，选择 Identity providers（身份提供程序）。
9. 选择 SAML。
10. 在 Provider name（提供商名称）中输入您的 SSO 提供商的名称。
11. 选择 Select file（选择文件），然后选择您在步骤 1 中下载的元数据 XML 文件。
12. 选择 Create provider（创建提供商）。

查看已共享的控制面板的数量

您可以使用 CloudWatch 控制台查看当前正在与其他人共享的 CloudWatch 控制面板的数量。

查看正在共享的控制面板的数量

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。

3. Dashboard sharing（共享控制面板）部分会显示共享的控制面板的数量。
4. 要查看共享的控制面板，请选择 Username and password（用户名和密码）和 Public dashboards（公有控制面板）下的 ***number*** dashboards shared（共享控制面板数量）。

查看您的哪些控制面板已共享

您可以使用 CloudWatch 控制台查看您的哪些控制面板当前正在与其他人共享。

查看正在共享哪些控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 在控制面板列表中，请参阅 Share（共享）列。包含此列中图标的控制面板当前正在共享。
4. 要查看与哪些用户共享控制面板，请选择控制面板名称，然后依次选择 Actions（操作）、Share dashboard（共享控制面板）。

   Share dashboard（共享控制面板）***dashboard name########***页面会显示控制面板的共享方式。如果需要，可以通过选择 Stop sharing（停止共享），停止共享控制面板。

停止共享一个或多个控制面板

您可以停止共享单个共享控制面板，或一次性停止共享所有共享控制面板。

停止共享单个控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择共享控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 选择 Stop sharing（停止共享）。
6. 在确认框中，选择 Stop sharing（停止共享）。

停止共享所有的共享控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。

3. 在 Dashboard sharing（控制面板共享）部分中，选择 Stop sharing all dashboards（停止共享所有控制面板）。
4. 在确认框中，选择 Stop sharing all dashboards（停止共享所有控制面板）。

查看共享控制面板权限并更改权限范围

如果要查看共享控制面板用户的权限，或更改共享控制面板权限的范围，请按照本节中的步骤进行操作。

查看共享控制面板的权限

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择共享控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 在 Resources（资源）下，选择 IAM Role（IAM 角色）。
6. 在 IAM 控制台中，选择所显示的策略。
7. （可选）要限制共享控制面板用户可以查看哪些告警，请选择 Edit policy（编辑策略）并将 `cloudwatch:DescribeAlarms` 权限从其当前位置移动到一个新 `Allow` 语句，该语句仅列出您希望共享控制面板用户查看的告警的 ARN。请参阅以下示例。

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": [
        "AlarmARN1",
        "AlarmARN2"
    ]
}
```

如果您执行此操作，请务必从当前策略的如下部分中删除 `cloudwatch:DescribeAlarms` 权限：

```
{
   "Effect": "Allow",
    "Action": [
        "cloudwatch:GetInsightRuleReport",
```

```
        "cloudwatch:GetMetricData",
        "cloudwatch:DescribeAlarms",
        "ec2:DescribeTags"
    ],
    "Resource": "*"
}
```

8. （可选）要限制共享控制面板用户可以查看的 Contributor Insights 规则的范围，请选择 Edit policy（编辑策略）将 `cloudwatch:GetInsightRuleReport` 从当前位置移动到一个新 `Allow` 语句，该语句仅列出您希望共享控制面板用户查看的 Contributor Insights 规则的 ARN。请参阅以下示例。

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:GetInsightRuleReport",
    "Resource": [
        "PublicContributorInsightsRuleARN1",
        "PublicContributorInsightsRuleARN2"
    ]
}
```

如果您执行此操作，请务必从当前策略的如下部分中删除 `cloudwatch:GetInsightRuleReport`：

```
{
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport",
                "cloudwatch:GetMetricData",
                "cloudwatch:DescribeAlarms",
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
```

允许与您共享的人员查看复合告警

在共享控制面板时，默认情况下，控制面板上的复合告警小组件对与您共享控制面板的人员不可见。要使复合告警小组件可见，您需要向控制面板共享策略添加 `DescribeAlarms: *` 权限。该权限如下所示：

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
}
```

Warning

上述策略语句允许访问账户中的所有告警。要缩小 `cloudwatch:DescribeAlarms` 的范围，您必须使用 Deny 语句。您可以向策略添加一个 Deny 语句，并指定要锁定的告警的 ARN。该拒绝语句应类似于以下内容：

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
},
{
    "Effect": "Deny",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": [
        "SensitiveAlarm1ARN",
        "SensitiveAlarm1ARN"
    ]
}
```

允许与您共享的人员查看日志表小组件

在共享控制面板时，默认情况下，控制面板上的 CloudWatch Logs Insights 小组件对与您共享控制面板的人员不可见。这会影响现存的 CloudWatch Logs Insights 小组件以及在共享该小组件后添加到控制面板的任何小组件。

如果您希望这些人员能够查看 CloudWatch Logs 小组件，则必须向 IAM 角色添加权限以进行控制面板共享。

允许与您共享控制面板的人员查看 CloudWatch Logs 小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。

3. 选择共享控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 在 Resources（资源）下，选择 IAM Role（IAM 角色）。
6. 在 IAM 控制台中，选择所显示的策略。
7. 选择 Edit policy（编辑策略）并添加以下语句。在新语句中，我们建议您仅指定要共享的日志组的 ARN。请参阅以下示例。

```
{
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetLogRecord",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "SharedLogGroup1ARN",
                "SharedLogGroup2ARN"
           ]
        },
```

8. 选择 Save Changes（保存更改）。

如果您的控制面板共享 IAM policy 已包含带 * 的五个权限作为资源，我们强烈建议您更改策略并仅指定您想要共享的日志组的 ARN。例如，如果这些权限的 Resource 部分如下所示：

```
"Resource": "*"
```

更改策略以仅指定要共享的日志组的 ARN，如下例所示：

```
"Resource": [
  "SharedLogGroup1ARN",
  "SharedLogGroup2ARN"
]
```

允许与您共享的人员查看自定义小组件

当您共享控制面板时，默认情况下控制面板上的自定义小组件对与您共享控制面板的人员不可见。这既会影响现存的自定义小组件，也会影响共享后添加到控制面板的任何自定义小组件。

如果您希望这些人员能够查看自定义小组件，您必须向 IAM 角色添加权限以进行控制面板共享。

允许与您共享控制面板的人员查看自定义小组件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择共享控制面板的名称。
4. 依次选择 Actions（操作）、Share dashboard（共享控制面板）。
5. 在 Resources（资源）下，选择 IAM Role（IAM 角色）。
6. 在 IAM 控制台中，选择所显示的策略。
7. 选择 Edit policy（编辑策略）并添加以下语句。在新语句中，我们建议您仅指定要共享的 Lambda 函数的 ARN。请参阅以下示例。

```
{
  "Sid": "Invoke",
  "Effect": "Allow",
  "Action": [
      "lambda:InvokeFunction"
  ],
  "Resource": [
    "LambdaFunction1ARN",
    "LambdaFunction2ARN"
  ]
 }
```

8. 选择 Save Changes（保存更改）。

如果您的控制面板共享 IAM policy 已包含带 * 的权限作为资源，我们强烈建议您更改策略并仅指定要共享的 Lambda 函数的 ARN。例如，如果这些权限的 `Resource` 部分如下所示：

```
"Resource": "*"
```

更改策略以仅指定要共享的自定义小组件的 ARN，如以下示例所示：

```
"Resource": [
  "LambdaFunction1ARN",
  "LambdaFunction2ARN"
]
```

使用实时数据

您可以选择指标小部件是否显示实时数据。实时数据是将在最后一分钟内发布的，尚未完全聚合的数据。

- 如果关闭实时数据，则仅显示过去至少一分钟的聚合期的数据点。例如，如果使用 5 分钟聚合期，12:35 的数据点将从 12:35 聚合到 12:40，并在 12:41 时显示。
- 如果启用实时数据，则在相应聚合间隔内发布任何数据后，立即显示最新数据点。每次刷新显示时，最新数据点都可能会随着该聚合期内的新数据发布而发生变化。如果您使用累积统计数据（例如总和或样本数），则使用实时数据可能会导致图形末尾出现下滑。

您可以选择为整个控制面板启用实时数据，也可以为控制面板上的单个小部件启用实时数据。

选择是否在整个控制面板上使用实时数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 要永久打开或关闭控制面板上所有小组件的实时数据，请执行以下操作：
   a. 选择 Actions (操作)、Settings (设置)、Bulk update live data (批量更新实时数据)。
   b. 选择 Live Data on (实时数据打开) 或 Live Data off (实时数据关闭)，然后选择 Set (设置)”。
4. 要临时覆盖每个小组件的实时数据设置，请选择 Actions（操作）。然后，在 Overrides（覆盖）下的 Live data（实时数据）旁边，请执行以下任一操作：
   - 选择 On (打开) 可临时为所有小部件启用实时数据。
   - 选择 OFF (关闭) 可临时关闭所有小组件的实时数据。
   - 选择 Do not override (不覆盖) 可保留每个小部件的实时数据设置。

选择是否在单个小部件上使用实时数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 选择小部件，然后选择 Actions (操作)、Edit (编辑)。
4. 选择 Graph options (图形选项) 选项卡。
5. 选中或清除 Live Data (实时数据) 下的复选框。

查看动画控制面板

您可以查看动画控制面板，该控制面板可回放随时间变化而捕获的 CloudWatch 指标数据。这可以帮助您在问题发生后查看趋势、进行演示或分析问题。

控制面板中的动画小组件包括折线小组件、堆叠面积小组件、数字小组件和 Metrics Explorer 小组件。饼图、条形图、文本小组件和日志小组件会显示在控制面板中，而不会以动画形式显示。

查看动画控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择控制面板的名称。
4. 选择 Actions（操作）、Replay dashboard（回放控制面板）。
5. （可选）默认情况下，启动动画时，动画会显示为滑动窗口。如果您希望动画显示为逐点动画，请在动画暂停时选择放大镜图标并重置缩放。
6. 要开启动画，请选择“Play（播放）”按钮。您还可以选择后退和前进按钮以移动到其他时间点。
7. （可选）要更改动画的时间窗口，请选择日历然后选择时间段。
8. 要更改动画的速度，请选择 Auto speed（自动速度），然后选择新速度。
9. 在完成后，选择 Exit animate（退出动画）。

将 CloudWatch 控制面板添加到收藏夹列表

在 CloudWatch 控制台中，您可以将控制面板、警报和日志组添加到收藏夹列表。您可以从导航窗格访问收藏夹列表。以下程序描述如何将控制面板添加到收藏夹列表。

将控制面板添加到收藏夹列表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 在控制面板列表中，选择您想要收藏的控制面板名称旁边的星号。

   - （可选）您还可以通过在列表中选择控制面板，然后选择控制面板名称旁边的星号，收藏某个控制面板。
4. 要访问收藏夹列表，请在导航窗格中选择 Favorites and recents（收藏夹和最近记录）。菜单中有两个列。一列包含您收藏的控制面板、警报和日志组，另一列包含您最近访问过的控制面板、警报和日志组。

> **Tip**
>
> 在 CloudWatch 控制台导航窗格中，您可以通过 Favorites and recents（收藏夹和最近记录）菜单收藏控制面板，以及警报和日志组。在 Recently visited（最近访问）列中，将鼠标悬停在您想要收藏的控制面板上，然后选择旁边的星号。

更改 CloudWatch 控制面板的时间段覆盖设置或刷新间隔

您可以指定如何保留或修改添加到此控制面板的图表的时间段设置。

将自动时间段或持久时间范围应用于小组件时，图表的总体时间范围可能会影响您设置的时间段。

- 如果时间范围为一天或更短，则不会更改时间段设置。
- 如果时间范围介于 1 天到 3 天，则设置为低于 5 分钟的时间段将更改为 5 分钟。
- 如果时间范围超过 3 天，则设置为低于 1 小时的时间段将更改为 1 小时。

以下步骤介绍了如何使用控制台更改时段覆盖选项。您也可以使用控制面板 JSON 结构中的 `periodOverride` 字段来更改这些选项。有关更多信息，请参阅 Dashboard Body Overall Structure。

更改时间段覆盖选项

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 选择操作。
3. 在 Period override (时间段覆盖) 下，选择以下选项之一：

   - 选择 Auto (自动)，让每个图表上的指标时间段自动适应控制面板的时间范围。

- 选择 Do not override (不要覆盖) 以确保始终遵循每个图表的时间段设置。
- 选择其他选项之一将导致添加到控制面板的图表始终适应该所选时间作为其时间段设置。

  当关闭控制面板或刷新浏览器后，Period override (时间段覆盖) 始终会还原为 Auto (自动)。无法保存不同的时间段覆盖设置。

您可以更改 CloudWatch 控制面板上的数据刷新频率。

更改控制面板刷新间隔

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在刷新选项菜单（右上角）上，选择 10 秒、1 分钟、2 分钟、5 分钟或 15 分钟。

更改 CloudWatch 控制面板的时间范围或时区格式

可以更改时间范围以显示几分钟、几小时、几天或几周的控制面板数据。您还可以更改时区格式以按 UTC 或本地时间格式显示控制面板数据。本地时间按计算机操作系统中指定的时区显示。

> **Note**
>
> 如果您创建的控制面板中的图表包含了 100 个或更多的高精度指标，我们建议您不要设置超过 1 小时的时间范围。有关更多信息，请参阅 高精度指标。

> **Note**
>
> 如果控制面板的时间范围短于其上用于小组件的时间段，则会发生以下情况：
>
> - 小组件会被修改以显示该小组件的一个完整时间段对应的数据量，即使该时间段长于控制面板时间范围，也是如此。这样可以确保图表上至少有一个数据点。
> - 该数据点时间段的开始时间会被向后调整，以确保至少可以显示一个数据点。

New console

更改控制面板时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在控制面板屏幕中，执行以下任一操作：
   - 在仪表板的上部区域中，选择一个预定义的时间范围。这些时间从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）或者 1w（1 周））。
   - 或者，您可以选择以下自定义时间范围选项之一：
     - 选择 Custom（自定义），然后选择 Relative（相对）选项卡。选择 1 分钟到 15 个月的时间范围。
     - 选择 Custom（自定义），然后选择 Absolute（绝对）选项卡。使用日历或文本字段指定时间范围。

> **Tip**
>
> 如果在聚合期设置为 Auto（自动）时更改图表的时间范围，则 CloudWatch 可能会更改时间段。要手动设置时间段，请选择 Actions（操作）下拉菜单，然后选择 Period override（时间段覆盖）。

更改控制面板时区格式

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在仪表板的上部区域中，选择自定义。
4. 在显示的框的右上角，从下拉菜单中选择 UTC（协调世界时）或 Local time（本地时间）。
5. 选择 应用。

Old console

更改控制面板时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在控制面板屏幕中，执行以下任一操作：
   - 在仪表板的上部区域中，选择一个预定义的时间范围。这些时间从 1 小时到 1 周（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。
   - 或者，您可以选择以下自定义时间范围选项之一：
     - 选择 custom（自定义）下拉菜单，然后选择 Relative（相对）选项卡。选择某个预定义的范围，其跨度从 1 分钟到 15 个月。
     - 选择 custom（自定义）下拉菜单，然后选择 Absolute（绝对）选项卡。使用日历或文本字段指定时间范围。

> **Tip**
>
> 如果在聚合期设置为 Auto（自动）时更改图表的时间范围，则 CloudWatch 可能会更改时间段。要手动设置时间段，请选择 Actions（操作）下拉菜单，然后选择 Period override（时间段覆盖）。

更改控制面板时区格式

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards（控制面板），然后选择一个控制面板。
3. 在控制面板屏幕的右上角，选择 Custom（自定义）下拉菜单。
4. 在显示的框的右上角，从下拉菜单中选择 UTC（协调世界时）或 Local timezone（本地时区）。

Amazon CloudWatch 指标

指标是关于您的系统性能的数据。预设情况下，许多服务都提供资源（例如，Amazon EC2 实例、Amazon EBS 卷和 Amazon RDS 数据库实例）的免费指标。您也可以对某些资源（例如 Amazon EC2 实例）启用详细监控，或发布您自己的应用程序指标。Amazon CloudWatch 可以加载您账户中的所有指标（您提供的 AWS 资源指标和应用程序指标）以进行搜索、绘制图表及设置告警。

指标数据的保留期限为 15 个月，这使您能够查看最新数据和历史数据。

要在控制台中绘制指标图表，您可以使用 CloudWatch Metrics Insights，这是一个高性能的 SQL 查询引擎，可用于实时识别所有指标中的趋势和模式。

内容

- 基本监控和详细监控
- 使用 CloudWatch Metrics Insights 查询您的指标
- 使用 Metrics Explorer 按标签和属性监控资源
- 使用指标流
- 查看可用的指标
- 绘制指标的图表
- 使用 CloudWatch 异常检测
- 使用指标数学
- 在图表中使用搜索表达式
- 获取指标的统计数据
- 发布 自定义指标

基本监控和详细监控

CloudWatch 提供两类监控：基本监控和详细监控。

很多 AWS 服务通过向 CloudWatch 发布一组默认指标来提供基本监控，而不会为客户产生费用。默认情况下，当您开始使用其中某一 AWS 服务 时，将自动启用基本监控。有关提供基本监控的服务的列表，请参阅 发布 CloudWatch 指标的 AWS 服务。

仅有一些服务提供详细监控。它还会产生费用。要将其用于 AWS 服务，您必须选择激活它。有关定价的更多信息，请参见 Amazon CloudWatch 定价。

详细监控选项因提供这种监控的服务而异。例如，Amazon EC2 详细监控提供了更频繁的指标，每隔一分钟发布一次，而不是 Amazon EC2 基本监控中每五分钟发布一次。针对 Amazon S3 和 Amazon Managed Streaming for Apache Kafka 进行详细监控，意味着指标会更精细。

在不同 AWS 服务中，详细监控也有不同名称。例如，在 Amazon EC2 中，它被称为详细监控；在 AWS Elastic Beanstalk 中，它被称为增强监控；而在 Amazon S3 中，它被称为请求指标。

针对 Amazon EC2 使用详细监控，可帮助您更好地管理 Amazon EC2 资源，以便您可以找到趋势并更快地采取措施。对于 Amazon S3，请求指标每隔一分钟可以使用一次，可帮助您快速识别运行问题，并针对其采取措施。在 Amazon MSK 上，当您启用 `PER_BROKER`、`PER_TOPIC_PER_BROKER` 或 `PER_TOPIC_PER_PARTITION` 级别监控时，您将获得可以提供更多可见性的额外指标。

下表列出了提供详细监控的服务。它还包括指向这些服务的文档的链接，这些文档说明了关于详细监控的更多信息，并提供了如何激活该功能的说明。

服务	文档
Amazon API Gateway	API Gateway 指标的维度
Amazon CloudFront	查看其他 CloudFront 分配指标
Amazon EC2	对实例启用或禁用详细监控
Elastic Beanstalk	增强型运行状况报告和监控
Amazon Kinesis Data Streams	增强的分片级指标
Amazon MSK	用于使用 CloudWatch 进行监控的 Amazon MSK 指标

服务	文档
Amazon S3	CloudWatch 中的 Amazon S3 请求指标
Amazon SES	使用 Amazon SES 事件发布功能收集 CloudWatch 详细监控指标。

此外，CloudWatch 还提供开箱即用的监测解决方案，其中包含更详细的指标和为某些 AWS 服务预创建的控制面板，如下表所示。

服务	功能文档
Lambda	Lambda Insights
Amazon ECS	适用于 Amazon ECS 的 Container Insights
Amazon EKS	适用于 Amazon EKS 和 Kubernetes 的 Container Insights

使用 CloudWatch Metrics Insights 查询您的指标

CloudWatch Metrics Insights 是一个功能强大的高性能 SQL 查询引擎，您可以使用它来大规模查询指标。您可以实时识别所有 CloudWatch 指标中的趋势和模式。

您还可以对返回单个时间序列的任何 Metrics Insights 查询设置告警。这对于创建告警以监视基础设施或应用程序实例集的聚合指标特别有用。只需创建一次告警，它就会随着实例集添加或删除资源而动态调整。

您可以使用 CloudWatch Metrics Insights 查询编辑器在控制台中执行 CloudWatch Metrics Insights 查询。您还可以通过运行 `GetMetricData` 或 `PutDashboard` 来使用 AWS CLI 或 AWS SDK 执行 CloudWatch Metrics Insights 查询。使用 CloudWatch Metrics Insights 查询编辑器运行的查询不收取任何费用。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

通过 CloudWatch Metris Insights 查询编辑器，您可以从各种预构建的示例查询中进行选择，也可以创建自己的查询。在创建查询时，您可以利用构建器视图浏览现有的指标和维度。或者，也可以利用编辑器视图手动编写查询。

您也可以使用自然语言创建 CloudWatch Metrics Insights 查询。要执行此操作，请询问或描述您要查找的数据。此 AI 辅助功能可按照您的提示生成查询，并逐行说明查询的工作原理。有关更多信息，请参阅使用自然语言生成与更新 CloudWatch Metrics Insights 查询。

借助 Metric Insights，您可以大规模运行查询。通过 GROUP BY 子句，您可以将指标实时分组为每个特定维度值的不同时间序列。Metrics Insights 查询包含排序依据功能，因此您可以使用 Metrics Insights 执行“前 N 个”类型的查询。例如，“前 N 个”类型的查询可以扫描账户中的数百万个指标，并返回 CPU 占用量最大的前 10 个实例。这有助于查明和修复应用程序中的延迟问题。

主题

- 构建查询
- Metrics Insights 查询组件和语法
- 为 Metrics Insights 查询创建告警
- 将 Metrics Insights 查询与指标数学配合使用
- 使用自然语言生成与更新 CloudWatch Metrics Insights 查询
- SQL 推理
- Metrics Insights 示例查询
- Metrics Insights 限制
- Metrics Insights 术语表
- Metrics Insights 问题排查

构建查询

您可以使用 CloudWatch 控制台、AWS CLI 或者 AWS SDK 来运行 CloudWatch Metrics Insights 查询。在控制台中运行的查询免费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

有关使用 AWS SDK 执行 Metrics Insights 查询的更多信息，请参阅 GetMetricData。

要使用 CloudWatch 控制台运行查询，请执行以下步骤：

使用 Metrics Insights 查询指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 选择 Query（查询）选项卡。
4. （可选）要运行预构建的示例查询，请选择 Add query（添加查询），然后选择要运行的查询。如果您对此查询满意，您可以跳过此过程的其余部分。或者，您可以选择 Editor（编辑器）来编辑示例查询，然后选择 Run（运行）以运行修改后的查询。
5. 要创建查询，您可以使用 Builder（构建器）视图、Editor（编辑器）视图，还可以使用两者的组合。您可以随时在两个视图之间切换，并在两个视图中查看正在进行的工作。

   在 Builder（构建器）视图中，您可以浏览并选择指标命名空间、指标名称、筛选条件、分组和排序选项。对于其中每个选项，查询构建器都会为您提供您环境中的可选项列表，供您选择。

   在 Editor（编辑器）视图中，您可以开始编写查询。输入时，编辑器会根据您当前已输入的字符提供建议。
6. 如果对查询感到满意，请选择 Run（运行）。
7. （可选）编辑已绘制图表查询的另一种方法是选择 Graphed metrics（已绘制图表指标）选项卡，然后选择 Details（详细信息）列中查询公式旁边的编辑图标。
8. （可选）要从图表中删除查询，请选择 Graphed metrics（已绘制图表指标），然后选择显示查询一行右侧的 X 图标。

Metrics Insights 查询组件和语法

CloudWatch Metrics Insights 语法如下。

```
SELECT FUNCTION(metricName)
FROM namespace | SCHEMA(...)
[ WHERE labelKey OPERATOR labelValue [AND ... ] ]
```

```
[ GROUP BY labelKey [ , ... ] ]
[ ORDER BY FUNCTION() [ DESC | ASC ] ]
[ LIMIT number ]
```

Metrics Insights 查询中可能的子句如下。关键字不区分大小写，但是指标名称、命名空间和维度等标识符需区分大小写。

SELECT

必需。指定用于聚合每个时间段的观测值的函数（由提供的时段确定）。还可以指定要查询的指标的名称。

FUNCTION 的有效值为 AVG、COUNT、MAX、MIN 和 SUM。

- AVG 计算查询匹配的观测值的平均值。
- COUNT 返回查询匹配的观测值的计数。
- MAX 返回查询匹配的观测值的最大值。
- MIN 返回查询匹配的观测值的最小值。
- SUM 计算查询匹配的观测值的总和。

FROM

必需。指定指标源。您可以指定包含要查询的指标的指标命名空间，也可以指定 SCHEMA 表函数。指标命名空间的示例包括："AWS/EC2"、"AWS/Lambda" 以及您为自定义指标创建的指标命名空间。

包括 / 或任何其他不是字母、数字或下划线的字符在内的指标命名空间必须放在双引号中。有关更多信息，请参阅 什么内容需要引号或转义字符？。

SCHEMA

一个可以在 FROM 子句中使用的可选表函数。使用 SCHEMA 将查询结果范围缩小到仅与维度列表完全匹配的指标或没有维度的指标。

如果您使用 SCHEMA 子句，它必须包含至少一个参数，并且第一个参数必须是被查询的指标命名空间。如果您仅用此命名空间参数指定 SCHEMA，则结果范围只限于没有任何维度的指标。

如果您使用其他参数指定 SCHEMA，命名空间参数之后的其他参数必须为标注键。标签键必须是维度名称。如果您指定了一个或多个标签键，则结果范围仅限于具有该确切维度集的指标。这些标签键的顺序无关紧要。

例如：

- SELECT AVG(CPUUtilization) FROM "AWS/EC2" 匹配 AWS/EC2 命名空间中的所有 CPUUtilization 指标（无论它们的维度如何），并且会返回单个聚合时间序列。
- SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2") 只匹配 AWS/EC2 命名空间中没有定义任何维度的 CPUUtilization 指标。
- SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId) 只匹配向 CloudWatch 报告且只有一个维度（即 CPUUtilization）的 InstanceId 指标。
- SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone) 只匹配从 AWS/ApplicationELB 向 CloudWatch 报告且只带有 LoadBalancer 和 AvailabilityZone 两个维度的 RequestCount 指标。

WHERE

可选。使用一个或多个标签键的特定标签值，将结果筛选为仅与指定表达式匹配的指标。例如，WHERE InstanceType = 'c3.4xlarge' 将结果筛选为仅 c3.4xlarge 实例类型，WHERE InstanceType != 'c3.4xlarge' 将结果筛选为除 c3.4xlarge 外的所有实例类型。

在监控账户中运行查询时，可以利用 WHERE AWS.AccountId 将查询结果限制到指定账户。例如，WHERE AWS.AccountId=444455556666 仅查询账户 444455556666 的指标。要将查询限制为仅限监控账户本身中的指标，请使用 WHERE AWS.AccountId=CURRENT_ACCOUNT_ID()。

标签值必须始终放在单引号中。

支持的运算符

WHERE 子句支持以下运算符：

- = 标签值必须与指定的字符串匹配。
- != 标签值不得与指定的字符串匹配。
- AND 指定的两个条件都必须为真才能匹配。您可以使用多个 AND 关键字来指定两个或更多条件。

GROUP BY

可选。将查询结果分组为多个时间序列，每个时间序列对应指定的一个或多个标签键的不同值。例如，使用 GROUP BY InstanceId 为每个 InstanceId 值返回不同的时间序列。使用 GROUP BY ServiceName, Operation 为 ServiceName 和 Operation 值的每个可能组合创建不同的时间序列。

通过 GROUP BY 子句，默认情况下，将使用 GROUP BY 子句中指定的标签序列将结果按字母升序排列。要更改结果的顺序，请在查询中添加 ORDER BY 子句。

在监控账户中运行查询时，您可以使用 GROUP BY AWS.AccountId 来按照结果的来源账户对结果进行分组。

> **Note**
>
> 如果某些匹配的指标不包含 GROUP BY 子句中指定的标签键，则会返回名为 Other 的空组。例如，如果指定 GROUP BY ServiceName, Operation 而且某些返回的指标不包括作为维度的 ServiceName，则这些指标将显示为将 Other 作为 ServiceName 的值。

ORDER BY

可选。如果查询返回多个时间序列，则指定返回的时间序列的使用顺序。该顺序基于您在 ORDER BY 子句中指定的 FUNCTION 找到的值。FUNCTION 用于计算每个返回的时间序列中的单个标量值，该值用于确定顺序。

还可以指定使用升序 ASC 或降序 DESC。如果您省略这一步，则默认为升序 ASC。

例如，添加 ORDER BY MAX() DESC 子句将按时间范围内观察到的最大数据点以降序对结果进行排序：这意味着具有最高最大数据点的时间序列会首先返回。

要在 ORDER BY 子句中使用的有效函数为 AVG()、COUNT()、MAX()、MIN() 和 SUM()。

如果您将 ORDER BY 子句与 LIMIT 子句配合使用，则生成的查询为“前 N 个”查询。ORDER BY 对于可能返回大量指标的查询也很有用，因为每个查询返回的时间序列不超过 500 个。如果查询匹配超过 500 个时间序列，并且您使用 ORDER BY 子句并对时间序列进行排序，则排序中前 500 个时间序列是返回的时间序列。

LIMIT

可选。将查询返回的时间序列数量限制为指定的值。您可以指定的最大值为 500，而不指定 LIMIT 的查询也可以返回不超过 500 个时间序列。

将 LIMITED 子句与 ORDER BY 子句配合使用，将获得“前 N 个”查询。

什么内容需要引号或转义字符？

在查询中，标签值必须始终放在单引号中。例如，SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet'。

包含字母、数字和下划线 (_) 以外的字符的指标命名空间、指标名称和标签键必须放在双引号中。例如，SELECT MAX("My.Metric")。

如果其中一个已包含双引号或单引号（如 Bytes"Input"），您必须用反斜杠转义每个引号，如 SELECT AVG("Bytes\"Input\"")。

如果指标命名空间、指标名称或标签键包含的词为 Metrics Insights 中的保留关键字，则这些词也必须放在双引号中。例如，如果您有名为 LIMIT 的指标，您可以使用 SELECT AVG("LIMIT")。将任何命名空间、指标名称或标签（即使不包含保留关键字）放在双引号中也有效。

有关保留关键字的完整列表，请参阅 保留关键字。

逐步构建丰富的查询

本节说明了如何逐步构建一个使用所有可能子句的完整示例。

我们从以下查询开始，该查询聚合了使用 LoadBalancer 和 AvailabilityZone 两个维度收集的所有 Application Load Balancer RequestCount 指标。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

现在，如果我们只想查看来自特定负载均衡器的指标，我们可以添加 WHERE 子句，将返回的指标限制为 LoadBalancer 维度的值为 app/load-balancer-1 的指标。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

前面的查询将此负载均衡器的所有可用区中的 RequestCount 指标聚合为一个时间序列。如果我们想看到每个可用区的不同时间序列，可以添加 GROUP BY 子句。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

接下来，我们可能希望对这些结果进行排序以首先查看最高值。以下 ORDER BY 子句按查询时间范围内每个时间序列报告的最大值以降序对时间序列进行排序：

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

```
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
```

最后，如果我们主要对“前 N 个”类型的查询感兴趣，我们可以使用 LIMIT 子句。这最后一个例子将结果限制为仅带有五个最高 MAX 值的时间序列。

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

跨账户查询示例

这些示例在 CloudWatch 跨账户可观测性中设置为监控账户的账户中运行时有效。

以下示例搜索源账户 123456789012 中的所有 Amazon EC2 实例并返回平均值。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId ='123456789012'
```

以下示例查询所有关联源账户中 AWS/EC2 的 CPUUtilization 指标，并按账户 ID 和实例类型对结果进行分组。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
GROUP BY AWS.AccountId, InstanceType
```

以下示例查询监控账户本身中的 CPUUtilization。

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = CURRENT_ACCOUNT_ID()
```

保留关键字

以下是 CloudWatch Metrics Insights 中的保留关键字。如果查询中的命名空间、指标名称或标签键中包含这些词中的任何一个，则必须将其放在双引号中。保留关键字不区分大小写。

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
 "ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
 "ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN"
 "ASINH" "ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE"
 "AVERAGE" "AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE"
 "BT" "BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N"
 "CAST" "CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
 "CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID"
 "COLUMN" "COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
 "CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
 "COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
 "CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
 "CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE"
 "DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
 "DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES"
 "DEL" "DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS"
 "DICTIONARY" "DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE"
 "DUAL" "DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING"
 "END_FRAME" "END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
 "EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
 "FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
 "FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
 "GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
```

```
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
 "HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE"
 "ILIKE" "IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
 "INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"
"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT"
 "INTERSECTION" "INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
"IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
 "JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
"JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS"
 "KILL" "KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
"LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
 "LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
"LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
 "LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
"LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY"
 "LT" "MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
"MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
 "MATERIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
"MCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
 "MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
"MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG"
 "MLOAD" "MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
"MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM"
 "MULTISET" "NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
"NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
 "NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
"NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
 "NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
"OCCURRENCES_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
 "OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
"OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE"
 "OPTIMIZER_COSTS" "OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
"OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
 "PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
"PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
 "PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
"PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
 "POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
"PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE"
 "PRIVILEGES" "PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
"PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
 "QUERYNO" "RADIANS" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
```

```
"RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
 "REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
"REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
 "RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
"REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
 "RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"
"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
 "ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
 "ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
 "SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL"
 "SELECT" "SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER"
 "SET" "SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT"
 "SOME" "SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SQLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
 "SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQRT" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
 "STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
 "SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"
 "SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN"
 "THRESHOLD" "TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION"
 "TRANSLATE" "TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"
 "UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"
 "UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF"
 "VARBINARY" "VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"
 "VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN"
 "WITHIN_GROUP" "WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"
```

为 Metrics Insights 查询创建告警

您可以为 Metrics Insights 查询创建告警。这有助于您获得跟踪多个资源的告警，而无需稍后更新。该查询可捕获新资源和发生变化的资源。例如，您可以创建一个告警来监视实例集的 CPU 利用率，该告警会自动评估您在创建告警后启动的新实例。

在为 CloudWatch 跨账户可观测性设置的监控账户中，Metris Insights 警报可以监控源账户中的资源和该监控账户本身中的资源。有关如何将警报查询限制在特定账户，以及如何按账户 ID 对结果进行分组的更多信息，请参阅 Metrics Insights 查询组件和语法 中的 `WHERE` 和 `GROUP BY` 部分。

目录

- 创建 Metrics Insights 告警
- 使用部分数据的案例

创建 Metrics Insights 告警

使用控制台为 Metrics Insights 查询创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 选择 Query（查询）选项卡。
4. （可选）要运行预构建的示例查询，请选择 Add query（添加查询），然后选择要运行的查询。或者，您可以选择 Editor（编辑器）来编辑示例查询，然后选择 Run（运行）以运行修改后的查询。
5. 要创建您自己的查询，您可以使用 Builder（构建器）视图、Editor（编辑器）视图或两者的组合。您可以随时在两个视图之间切换，并在两个视图中查看正在进行的工作。

   在 Builder（构建器）视图中，您可以浏览并选择指标命名空间、指标名称、筛选条件、分组和排序选项。对于其中每个选项，查询构建器都会为您提供您环境中的可选项列表，供您选择。

   在 Editor（编辑器）视图中，您可以开始编写查询。输入时，编辑器会根据您当前已输入的字符提供建议。

   > **Important**
   >
   > 要为 Metrics Insights 查询设置告警，该查询必须返回单个时间序列。如果该查询包含 GROUP BY 语句，则 GROUP BY 语句必须封装在指标数学表达式中，该表达式仅返回一个时间序列作为其最终结果。

6. 如果对查询感到满意，请选择 Run（运行）。
7. 选择创建警报。
8. 在条件下面，指定以下内容：

   a. 对于 Whenever ***metric*** is（每当指标），指定指标是必须大于、小于还是等于阈值。在于... 下面，指定阈值。

   b. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

      要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。

   c. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
9. 选择下一步。
10. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

    要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

    要让警报不发送通知，请选择删除。
11. 要让告警执行 Auto Scaling、EC2 或 Systems Manager 操作，请选择相应的按钮，然后选择告警状态和要执行的操作。告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。
12. 在完成后，选择下一步。
13. 输入警报的名称和说明。名称只能包含 ASCII 字符。然后选择下一步。
14. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

使用 AWS CLI 为 Metrics Insights 查询创建告警

- 使用 `put-metric-alarm` 命令并在 `metrics` 参数中指定 Metrics Insights 查询。例如，以下命令将设置一个告警，如果任何实例的 CPU 利用率超过 50%，该告警将进入 ALARM 状态。

```
aws cloudwatch put-metric-alarm --alarm-name Metrics-Insights-alarm --
evaluation-periods 1 --comparison-operator GreaterThanThreshold --metrics
 '[{"Id":"m1","Expression":"SELECT MAX(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
 InstanceId)", "Period":60}]' --threshold 50
```

使用部分数据的案例

如果用于告警的 Metrics Insights 查询匹配的指标超过 10,000 个，则根据该查询找到的前 10,000 个指标评估告警。这意味着根据部分数据评估告警。

您可以使用以下方法来确定 Metrics Insights 告警目前是否正在根据部分数据评估其告警状态：

- 在控制台中，如果您选择一个告警来查看 Details（详细信息）页面，则该页面上会显示 Evaluation warning: Not evaluating all data（评估警告：未评估所有数据）消息。
- 当您使用 describe-alarms AWS CLI 命令或 DescribeAlarms API 时，您会在 `EvaluationState` 字段中看到值 `PARTIAL_DATA`。

当 Amazon EventBridge 进入部分数据状态时，告警还会将事件发布到 Amazon EventBridge，这样您就可以创建 EventBridge 规则来监视这些事件。在这些事件中，`evaluationState` 字段的值为 `PARTIAL_DATA`。示例如下：

```
{
    "version": "0",
    "id": "12345678-3bf9-6a09-dc46-12345EXAMPLE",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-11-08T11:26:05Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:my-alarm-name"
    ],
    "detail": {
        "alarmName": "my-alarm-name",
        "state": {
```

```
            "value": "ALARM",
            "reason": "Threshold Crossed: 3 out of the last 3 datapoints [20000.0
 (08/11/22 11:25:00), 20000.0 (08/11/22 11:24:00), 20000.0 (08/11/22 11:23:00)] were
 greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2022-11-08T11:26:05.399+0000\",\"startDate\":\"2022-11-08T11:23:00.000+0000\",
\"period\":60,\"recentDatapoints\":[20000.0,20000.0,20000.0],\"threshold\":0.0,
\"evaluatedDatapoints\":[{\"timestamp\":\"2022-11-08T11:25:00.000+0000\",\"value
\":20000.0}]}",
            "timestamp": "2022-11-08T11:26:05.401+0000",
            "evaluationState": "PARTIAL_DATA"
        },
        "previousState": {
            "value": "INSUFFICIENT_DATA",
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2022-11-08T11:25:51.227+0000"
        },
        "configuration": {
            "metrics": [
                {
                    "id": "m2",
                    "expression": "SELECT SUM(PartialDataTestMetric) FROM
 partial_data_test",
                    "returnData": true,
                    "period": 60
                }
            ]
        }
    }
}
```

如果告警查询包含的 GROUP BY 语句最初返回 500 个以上的时间序列，则将根据该查询找到的前 500 个时间序列对告警进行评估。但是，如果您使用的是 ORDER BY 子句，则将对该查询找到的所有时间序列进行排序，并根据您的 ORDER BY 子句，将具有最高值或最低值的前 500 个时间序列用于评估告警。

将 Metrics Insights 查询与指标数学配合使用

您可以使用 Metrics Insights 查询作为指标数学函数的输入。有关指标数学的更多信息，请参阅 使用指标数学。

不包括 GROUP BY 子句的 Metrics Insights 查询将返回单个时间序列。因此，其返回的结果可以与以单个时间序列作为输入的任何指标数学函数一起使用。

包括 GROUP BY 子句的 Metrics Insights 查询将返回多个时间序列。因此，其返回的结果可以与以时间序列数组作为输入的任何指标数学函数一起使用。

例如，以下查询以时间序列数组的形式返回为区域中每个存储桶下载的总字节数：

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
```

在控制台或 GetMetricData 操作中的图表上，此查询的结果为 q1。此查询返回以字节为单位的结果，因此如果您想查看 MB 单位的结果，可以使用以下数学函数：

```
q1/1024/1024
```

使用自然语言生成与更新 CloudWatch Metrics Insights 查询

> **Note**
>
> 此功能在 CloudWatch 的美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和亚太地区（东京）已正式发布。

CloudWatch 支持自然语言查询功能，以帮助您生成和更新 CloudWatch Metrics Insights 和 CloudWatch Logs Insights 查询。

通过此功能，您可以用通俗易懂的英语询问或描述要查找的 CloudWatch 数据。自然语言功能可根据您输入的提示生成查询，并逐行说明查询的工作原理。您也可以更新查询以进一步调查数据。

您可以根据环境输入各种提示，例如“Which Amazon Elastic Compute Cloud instance has the highest network out?”（哪个 Amazon Elastic Compute Cloud 实例的网络输出量最高？）以及“Show me the top 10 Amazon DynamoDB Tables by consumed reads”（展示读取使用量前 10 的 Amazon DynamoDB 表）等。

要生成具有此功能的 CloudWatch Metrics Insights 查询，请在构建器或编辑器视图中打开 CloudWatch Metrics Insights 查询编辑器，然后选择生成查询。

> **Important**
>
> 要使用自然语言查询功能，您必须使用 CloudWatchFullAccess、CloudWatchReadOnlyAccess、CloudWatchFullAccessV2、AdministratorAcc 或 ReadOnlyAccess 策略。
> 您也可以将 `cloudwatch:GenerateQuery` 操作包含在新的或现有的客户托管策略或内联策略中。

示例查询

本节中的示例描述了如何使用自然语言功能生成和更新查询。

> **Note**
>
> 有关 CloudWatch Metrics Insights 查询编辑器和语法的更多信息，请参阅 CloudWatch Metrics Insights 查询组件和语法。

示例：生成自然语言查询

要使用自然语言来生成查询，请输入提示并选择生成新查询。此示例显示了一个执行基本搜索的查询。

提示

在以下示例中，该提示指示的功能可搜索占用最多读取容量的前 10 个 DynamoDB 表格。

```
Show top 10 DynamoDB Tables by consumed reads
```

查询

以下是自然语言功能根据提示生成的查询示例。请注意在查询之前的注释中，提示是如何显示的。查询完成后，您可以阅读有关查询工作原理的说明。

```
# Show top 10 DynamoDB Tables by consumed reads
SELECT SUM("ConsumedReadCapacityUnits")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
```

```
# This query selects the sum of consumed read capacity units for each DynamoDB table,
 groups the results by table name, orders the results from highest to lowest read
 capacity consumption, and limits the results to the top 10 tables.
```

> **Note**
>
> 要关闭提示和有关查询工作原理的说明显示，请使用编辑器中的齿轮图标。

示例：更新自然语言查询

您可以通过编辑初始提示，然后选择更新查询，来更新查询。

更新版提示

以下示例显示了先前提示的更新版本。现在，该提示不再搜索占用读取容量最多的前 10 个 DynamoDB 表格，而是按返回的字节数对结果进行排序。

```
Sort by bytes returned instead
```

更新版查询

以下是更新版查询的示例：请注意在更新版查询之前的注释中，更新版提示是如何显示的。查询完成后，您可以阅读有关原始查询是如何更新的说明。

```
# Sort by bytes returned instead
SELECT SUM("ReturnedBytes")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query modifies the original query to select the sum of returned bytes instead
 of consumed read capacity units, and orders the results from highest to lowest sum of
 returned bytes, limiting the results to the top 10 tables.
```

选择不使用您的数据来改善服务

您为训练 AI 模型和生成相关查询而提供的自然语言提示数据仅用于提供和维护您的服务。这些数据可用于提高 CloudWatch Metrics Insights 的质量。您的信任、隐私和内容的安全性是我们最重视的问题。有关更多信息，请参阅 AWS Service Terms 和 AWS responsible AI policy。

通过创建 AI 服务选择退出策略，您可以选择不将您的内容用于开发自然语言查询或提高自然语言查询的质量。要选择退出对所有 CloudWatch AI 功能（包括查询生成功能）进行的数据收集，您必须为 CloudWatch 创建一个选择退出策略。有关更多信息，请参阅《AWS Organizations 用户指南》中的 AI 服务选择退出策略。

SQL 推理

CloudWatch Metrics Insights 使用多种机制来推断给定 SQL 查询的意图。

主题

- 时间分段
- 字段投影
- ORDER BY 全局聚合

时间分段

根据请求的时段，查询生成的时间序列数据点将累计到时间段中。要在标准 SQL 中聚合值，必须定义一个明确的 GROUP BY 子句，以便一起收集给定时段的所有观测值。由于这是查询时间序列数据的标准方式，CloudWatch Metrics Insights 可推断时间分段，而无需明确表达 GROUP BY 子句。

例如，当执行时间为 1 分钟的查询时，属于该分钟的所有观测值，直到下一分钟（排除）为止，都将累计到该时间段的开始时间。这使得 Metrics Insights SQL 语句更加简洁明了。

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

上一个查询返回单个时间序列（时间戳值对），表示所有 Amazon EC2 实例的平均 CPU 使用率。假设请求的时段为 1 分钟，则返回的每个数据点代表在特定 1 分钟间隔（包括开始时间，不包括结束时间）内测量的所有观测值的平均值。与特定数据点相关的时间戳是时间段的开始时间

字段投影

Metrcis Insights 查询始终返回时间戳投影。您不需要在 SELECT 子句中指定时间戳列来获取每个相应数据点值的时间戳。有关如何计算时间戳的详细信息，请参阅 时间分段。

使用 GROUP BY 时，还可以在结果中推断和投影每个组名称，以便您可以对返回的时间序列进行分组。

```
SELECT AVG(CPUUtilization)
```

```
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

上一个查询将返回每个 Amazon EC2 实例的时间序列。每个时间序列都在实例 ID 的值之后进行标注。

ORDER BY 全局聚合

使用 ORDER BY 时、FUNCTION() 推断要按哪个聚合函数排序（查询指标的数据点值）。聚合操作在查询的时间窗口中对每个单独时间序列的所有匹配数据点执行。

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

上一个查询返回每个 Amazon EC2 实例的 CPU 使用率，并将结果集限制为 10 个条目。结果根据请求的时间窗口内各个时间序列的最大值进行排序。ORDER BY 子句在 LIMIT 之前应用，以便根据 10 个以上的时间序列计算排序。

Metrics Insights 示例查询

本节包含有用的 CloudWatch Metrics Insights 查询的示例，您可以直接复制和使用，或在查询编辑器中复制并修改。控制台中已经提供一些示例，您可以通过选择 Metrics（指标）视图中的 Add query（添加查询）来访问它们。

Application Load Balancer 示例

所有负载均衡器的请求总数

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

前 10 大最活跃的负载均衡器

```
SELECT MAX(ActiveConnectionCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
```

```
LIMIT 10
```

AWS API 使用示例

按账户中的调用次数划分，排名前 20 的 AWS API

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

按调用排序的 CloudWatch API

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB 示例

按消耗的读取数排序的前 10 个表

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

按返回的字节数排序的前 10 个表

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

按用户错误排序的前 10 个表

```
SELECT SUM(UserErrors)
```

```
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store 示例

按写入字节数排序的前 10 个 Amazon EBS 卷

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Amazon EBS 卷的平均写入时间

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 示例

按最高排序的 EC2 实例 CPU 使用率

```
SELECT AVG(CPUUtilization)
  FROM SCHEMA("AWS/EC2", InstanceId)
  GROUP BY InstanceId
  ORDER BY AVG() DESC
```

整个机群的平均 CPU 使用率

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

按最高 CPU 使用率排序的前 10 个实例

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX() DESC
LIMIT 10
```

在这种情况下，CloudWatch 代理正在根据应用程序收集 **CPUUtilization** 指标。此查询筛选特定应用程序名称的此指标的平均值。

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service 示例

所有 ECS 集群的平均 CPU 使用率

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

按内存利用率排序的前 10 个集群

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

按 CPU 使用率排序的前 10 项服务

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

按运行任务排序的前 10 项服务 (Container Insights)

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights 示例

所有 EKS 集群的平均 CPU 使用率

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

按节点 CPU 使用率排序的前 10 个集群

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

按 pod 内存利用率排序的前 10 个集群

```
SELECT AVG(pop_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

按 CPU 使用率排序的前 10 个节点

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
ORDER BY AVG() DESC LIMIT 10
```

按内存利用率排序的前 10 个 pod

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge 示例

按调用排序的前 10 条规则

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

按失败调用排序的前 10 条规则

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

按匹配规则排序的前 10 条规则

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis 示例

按写入字节数排序的前 10 个流

```
SELECT SUM("PutRecords.Bytes")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY SUM() DESC LIMIT 10
```

按流中最早的项目排序的前 10 个流

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda 示例

按调用次数排序的 Lambda 函数

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
```

按最长运行时排序的前 10 个 Lambda 函数

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

按错误计数排序的前 10 个 Lambda 函数

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs 示例

按传入事件排序的前 10 个日志组

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

按写入字节数排序的前 10 个日志组

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS 示例

按最高 CPU 使用率排序的前 10 个 Amazon RDS 实例

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

按写入数排序的前 10 个 Amazon RDS 集群

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service 示例

按存储桶排序的平均延迟

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

按已下载字节数排序的前 10 个存储桶

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon Simple Notification Service 示例

SNS 主题发布的消息总数

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

按发布消息排序的前 10 个主题

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
```

```
LIMIT 10
```

按消息传送失败排序的前 10 个主题

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS 示例

按可见消息数量排序的前 10 个队列

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

前 10 个最活跃的队列

```
SELECT SUM(NumberOfMessagesSent)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
LIMIT 10
```

按最早消息的时间排序的前 10 个队列

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights 限制

CloudWatch Metrics Insights 目前有以下限制：

- 目前，您只能查询最近三个小时的数据。

- 单个查询可处理的指标不超过 10,000 个。这意味着，如果 SELECT、FROM 和 WHERE 子句匹配 10,000 个以上的指标，查询只处理找到的前 10,000 个指标。
- 单个查询可返回的时间序列不超过 500 个。这意味着，如果查询返回 500 个以上的指标，则查询结果中并非所有指标都将返回。如果您使用的是 ORDER BY 子句，则将对正在处理的所有指标进行排序，并根据您的 ORDER BY 子句，具有前 500 个最高值或最低值的指标将返回。

  如果您不包括 ORDER BY 子句，则无法控制哪 500 个匹配的指标将返回。
- 每个区域最多可以有 200 个 Metrics Insights 警报。
- Metrics Insights 不支持高分辨率数据，即以小于一分钟的粒度报告的指标数据。如果您请求高分辨率数据，该请求不会失败，但输出会以一分钟的粒度聚合。
- 每个 GetMetricData 操作只能有一个查询，但是您可以在控制面板中有多个小组件，每个小组件都包含一个查询。

Metrics Insights 术语表

标签

在 Metrics Insights 中，标签是一个键值对，用于确定查询范围以返回特定数据集，或者定义将查询结果分为单独时间序列的标准。标签键与 SQL 中的列名称类似。目前，标签必须是 CloudWatch 指标维度。

观测值

观测值是指在给定时间为给定指标记录的值。

Metrics Insights 问题排查

结果包括“其他”，但未将其作为维度

这意味着查询包含 GROUP BY 子句，该子句指定查询返回的某些指标中未使用的标签键。在这种情况下，会返回名为 `Other` 的空组。不包含该标签键的指标可能是聚合指标，这些指标返回该标签键的所有值之间聚合的值。

例如，假设我们有以下查询：

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

如果某些返回的指标不作为维度包括 ServiceName，则这些指标将显示为将 Other 作为 ServiceName 的值。

要防止在结果中看到“其他”，请在您的 FROM 子句中使用 SCHEMA，如以下示例所示：

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
GROUP BY Operation, ServiceName
```

这将返回的结果限制为同时具有 Operation 和 ServiceName 维度的指标。

我的图表中最早的时间戳的指标值比其他时间戳的指标值低

CloudWatch Metrics Insights 目前仅支持最近三个小时的数据。当您使用大于 1 分钟的时间段绘制图表时，可能会出现最早的数据点与预期值不同的情况。这是因为 Metrics Insights 查询仅会返回最近三小时的数据。在这种情况下，查询中最早的数据点仅会返回最近三小时边界内测量的观测值，而不是返回该数据点时间段内的所有观测值。

使用 Metrics Explorer 按标签和属性监控资源

Metrics Explorer 是一个基于标签的工具，您可以使用它按标签和资源属性筛选、聚合和可视化指标，从而增强对服务的监控。这样，您可以灵活、动态地排查问题，从而能够一次创建多个图表并使用它们构建应用程序运行状况控制面板。

Metrics Explorer 可视化会动态变化，如果在创建 Metrics Explorer 小组件并将其添加到 CloudWatch 控制面板之后创建了匹配的资源，则新资源将自动显示在 Metrics Explorer 小组件中。

例如，如果您的所有 EC2 生产实例都具有 **`production`** 标签，则您可以使用 Metrics Explorer 筛选和聚合所有这些实例的指标，以了解其运行状况和性能。如果稍后创建了具有匹配标签的新实例，则该实例会自动添加到 Metrics Explorer 小组件中。

> **Note**
>
> Metrics Explorer 提供时间点体验。已终止的资源或不再以您指定的属性或标签存在的资源不会显示在可视化中。但是，您仍然可以在 CloudWatch 指标视图中找到这些资源的指标。

使用 Metrics Explorer，您可以选择如何聚合资源中与条件匹配的指标，以及是在单个图表中显示这些指标，还是在一个 Metrics Explorer 小组件中的不同图表上显示它们。

Metrics Explorer 包括您可以一键查看有用的可视化图表的模板，您还可以扩展这些模板以创建完全自定义的 Metrics Explorer 小组件。

Metrics Explorer 支持 AWS 发布的指标和由 CloudWatch 代理发布的 EC2 指标，包括内存、磁盘和 CPU 指标。若要使用 Metrics Explorer 查看由 CloudWatch 代理发布的指标，您可能需要更新您的 CloudWatch 代理配置文件。有关更多信息，请参阅 适合 Metrics Explorer 的 CloudWatch 代理配置

若要使用 Metrics Explorer 创建可视化图表并选择性地将其添加到控制面板，请按照下列步骤操作。

使用 Metrics Explorer 创建可视化图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格上，选择 Explorer。
3. 请执行以下操作之一：
   - 若要使用模板，请在当前显示 Empty Explorer（空 Explorer）的框中将其选中。

     根据不同模板，Explorer 可能会立即显示指标图表。如果未显示，请在 From（从）框中选择一个或多个标签或属性，随后应会显示数据。如果未显示数据，请使用页面顶部的选项将图表中显示的时间范围扩大。
   - 若要创建自定义可视化，请在 Metrics（指标）下，选择服务的单个指标或所有可用指标。

     选择指标之后，您可以选择性地重复此步骤添加更多指标。
4. 对于选定的每个指标，CloudWatch 会在指标名称后面显示其将随即使用的统计数据。若要更改此统计数据，请选择统计数据名称，然后选择您想要的统计数据。
5. 在 From（从）下，选择标签或资源属性来筛选结果。

   执行此操作后，您可以选择性地重复此步骤以选择更多标签或资源属性。

   如果您选择同一属性的多个值（如两个 EC2 实例类型），则 Explorer 将显示与任一选定属性匹配的所有资源。Explorer 将此选择视为 OR 操作。

   如果您选择了不同的属性或标签，例如 **`Production`** 标签和 M5 实例类型，则仅显示与所有这些选择匹配的资源。Explorer 将此选择视为 AND 操作。
6. （可选）对于 Aggregate by（聚合依据），选择要用于聚合指标的统计数据。然后，在 for（聚合方式）旁边，选择如何聚合列表中的指标。您可以将当前显示的所有资源聚合在一起，也可以按单个标签或资源属性聚合。

   根据您选择的聚合方式，结果可能是单个时间序列或多个时间序列。

7. 在 Split by（拆分方式）下，您可以选择将具有多个时间序列的单个图表拆分为多个图表。可以按您在 Split by（拆分方式）下选择的多种标准进行拆分。
8. 在 Graph options（绘制选项图形）下，您可以通过更改周期、图表类型、图例放置和布局来优化图表。
9. 若要将此可视化作为小组件添加到 CloudWatch 控制面板，请选择 Add to dashboard（添加到控制面板）。

适合 Metrics Explorer 的 CloudWatch 代理配置

要使 Metrics Explorer 能够发现 CloudWatch 代理发布的 EC2 指标，请确保 CloudWatch 代理配置文件包含以下值：

- 在 `metrics` 部分，请确保 `aggregation_dimensions` 参数包含 `["InstanceId"]`。其还可以包含其他维度。
- 在 `metrics` 部分，请确保 `append_dimensions` 参数包含 `{"InstanceId":"${aws:InstanceId}"}` 行。其还可以包含其他行。
- 在 `metrics` 部分（`metrics_collected` 部分中），检查您希望 Metrics Explorer 发现的每种资源类型所对应的部分，例如 `cpu`、`disk` 和 `memory` 部分。确保所有这些部分均包含 `"resources": [ "*"] line.`。
- 在 `cpu` 部分（`metrics_collected` 部分中），请确保具有 `"totalcpu": true` 行。
- 您必须为 CloudWatch 代理收集的指标使用原定设置的 `CWAgent` 命名空间，而不是自定义命名空间。

根据上一个列表中的设置，CloudWatch 代理会发布磁盘、CPU 和其它资源的聚合指标，这些指标可在 Metrics Explorer 中为使用它的所有实例标绘出来。

这些设置会重新发布您之前设置为以多个维度发布的指标，从而增加您的指标成本。

有关编辑 CloudWatch 代理配置文件的更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

使用指标流

您可以使用指标流持续地将 CloudWatch 指标流式传输到您所选的目标位置，实现近实时传送和低延迟。支持的目标位置包括 AWS 目标位置（例如 Amazon Simple Storage Service）和多个第三方服务提供商目标位置。

CloudWatch 指标流有 3 种主要使用场景：

- 使用 Firehose 进行自定义设置 – 创建指标流并将其导向 Amazon Data Firehose 传输流，该流会将您的 CloudWatch 指标传送到所需位置。您可以将它们流式传输到诸如 Amazon S3 之类的数据湖，或传输到 Firehose 支持的任何目标或端点，包括第三方提供商。原生支持 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0 格式，或者您可以在 Firehose 传输流中配置转换，将数据转换为其他格式，例如 Parquet。借助指标流，您可以持续更新监控数据，或者将此 CloudWatch 指标数据与账单和性能数据相结合，以创建丰富的数据集。然后，您可以使用 Amazon Athena 等工具深入了解成本优化、资源性能和资源利用率。
- S3 快速设置 – 通过快速设置过程流式传输到 Amazon Simple Storage Service。默认情况下，CloudWatch 会创建流所需的资源。支持的格式为 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0。
- AWS 合作伙伴快速设置 – CloudWatch 会为某些第三方合作伙伴提供快速设置功能体验。您可以通过第三方服务提供商，使用 CloudWatch 流式数据监控和分析您的应用程序并排查应用程序问题。当您使用合作伙伴快速设置工作流时，您只需要提供目标 URL 和目标的 API 密钥，CloudWatch 将负责其余的设置。合作伙伴快速设置适用于以下第三方提供商：

  - Datadog
  - Dynatrace
  - New Relic
  - Splunk Observability Cloud
  - SumoLogic

您可以流式传输所有 CloudWatch 指标，也可以使用筛选条件仅流式传输指定指标。每个指标流最多可包含 1000 个包括或排除指标命名空间或特定指标的筛选条件。单个指标流只能包括或排除筛选条件，不能同时既包括又排除。

创建指标流后，如果创建了与现有筛选条件匹配的新指标，则新指标将自动包含在指标流中。

每个账户或每个区域的指标流数量没有限制，正在流式传输的指标的更新数量也没有限制。

每个指标流可以使用 JSON、OpenTelemetry 1.0.0 或 OpenTelemetry 0.7.0 格式。您可以随时编辑指标流的输出格式，例如从 OpenTelemetry 0.7.0 升级到 OpenTelemetry 1.0.0。有关输出格式的更多信息，请参阅 指标流输出格式。

对于监控账户中的指标流，您可以选择是否包含与该监控账户关联的源账户的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

指标流始终包括 Minimum、Maximum、SampleCount 和 Sum 统计数据。您也可以选择包括其他统计数据，另外收费。有关更多信息，请参阅 可以流式传输的统计数据。

指标流定价基于指标更新的数量。对于用于指标流的传输流的 Firehose，也会产生费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

主题

- 设置指标流
- 可以流式传输的统计数据
- 指标流操作和维护
- 使用 CloudWatch 指标监控您的指标流
- CloudWatch 和 Firehose 之间的信任关系
- 指标流输出格式
- 故障排除

设置指标流

按照以下部分中的步骤设置 CloudWatch 指标流。

创建指标流后，指标数据显示在目标位置所需的时间取决于对 Firehose 传输流配置的缓冲设置。缓冲以最大负载大小或最长等待时间表示，以先达到者为准。如果缓冲设置为最小值（60 秒、1MB），则如果所选 CloudWatch 命名空间具有活动指标更新，则预期延迟将在 3 分钟内。

在 CloudWatch 指标流中，每分钟发送一次数据。数据可能无序到达最终目标位置。指定命名空间中的所有指定的指标都在指标流中发送，但时间戳超过两天的指标除外。

对于您流式传输的指标名称和命名空间的每个组合，该指标名称和命名空间的所有维度组合都将被流式传输。

对于监控账户中的指标流，您可以选择是否包含与该监控账户关联的源账户的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

要创建及管理指标流，您必须登录具有 CloudWatchFullAccess 策略和 `iam:PassRole` 权限的账户，或具有以下所列权限的账户：

- `iam:PassRole`
- `cloudwatch:PutMetricStream`

- cloudwatch:DeleteMetricStream
- cloudwatch:GetMetricStream
- cloudwatch:ListMetricStreams
- cloudwatch:StartMetricStreams
- cloudwatch:StopMetricStreams

如果您要让 CloudWatch 设置指标流所需的 IAM 角色，您还必须具有 iam:CreateRole 和 iam:PutRolePolicy 权限。

> **Important**
>
> 具有 cloudwatch:PutMetricStream 的用户即使不具有 cloudwatch:GetMetricData 权限也可以访问正在流式传输的 CloudWatch 指标数据。

主题

- 使用 Firehose 自定义设置
- 使用 Amazon S3 快速设置
- 合作伙伴快速设置

使用 Firehose 自定义设置

使用此方法创建指标流并将其导向 Amazon Data Firehose 传输流，该流会将您的 CloudWatch 指标传送到所需位置。您可以将它们流式传输到诸如 Amazon S3 之类的数据湖，或传输到 Firehose 支持的任何目标或端点，包括第三方提供商。

原生支持 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0 格式，或者您可以在 Firehose 传输流中配置转换，将数据转换为其他格式，例如 Parquet。借助指标流，您可以持续更新监控数据，或者将此 CloudWatch 指标数据与账单和性能数据相结合，以创建丰富的数据集。然后，您可以使用 Amazon Athena 等工具深入了解成本优化、资源性能和资源利用率。

您可以使用 CloudWatch 控制台、AWS CLI、AWS CloudFormation 或 AWS Cloud Development Kit (AWS CDK) 设置指标流。

您用于指标流的 Firehose 传输流必须位于指标流设置所在的同一账户和区域中。要实现跨区域功能，您可以将 Firehose 传输流配置为流式传输到位于不同账户或不同区域中的最终目标位置。

CloudWatch 控制台

本部分介绍如何通过 CloudWatch 控制台使用 Firehose 设置指标流。

要使用 Firehose 设置自定义指标流

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标）、Streams（流）。然后选择 Create metric stream（创建指标流）。
3. （可选）如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，您可以选择是否包含来自此指标流中关联源账户的指标。要包含来自源账户的指标，请选择 Include source account metrics（包含源账户指标）。
4. 选择使用 Firehose 自定义设置。
5. 对于选择 Kinesis Data Firehose 流，请选择要使用的 Firehose 传输流。必须位于同一账户中。此选项的默认格式为 OpenTelemetry 0.7.0，但您可以在此过程的后续步骤中更改格式。

   然后在选择您的 Firehose 传输流下选择要使用的 Firehose 传输流。
6. （可选）您可以选择选择现有服务角色来使用现有 IAM 角色，而不必让 CloudWatch 为您创建新角色。
7. （可选）要更改方案的默认输出格式，请选择 Change output format（更改输出格式）。支持的格式为 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0。
8. 对于要进行流式传输的指标，请选择所有指标或选择指标。

   如果您选择所有指标，则该账户的所有指标都将包含在流中。

   请仔细考虑是否要流式传输所有指标，因为流式传输的指标越多，您的指标流费用就越高。

   如果选择的是选择指标，请执行以下操作中的一项：

   - 要流式传输大多数指标命名空间，请选择排除，然后选择要排除的命名空间或指标。在排除中指定命名空间时，可以选择该命名空间中要排除的一些特定指标。如果选择排除某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将被排除。
   - 要在指标流中仅包含少数指标命名空间或指标，请选择包含，然后选择要包含的命名空间或指标。如果选择包含某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将包含在内。
9. （可选）要流式传输其中除 Minimum、Maximum、SampleCount 和 Sum 外的一些指标的其他统计数据，请选择添加其他统计数据。要么选择 Add recommended metrics（添加推荐的指标）添

加一些常用的统计数据，要么手动选择要针对其流式传输额外统计数据的命名空间和指标名称。接下来，选择要流式传输的额外统计数据。

然后，要选择另一组指标，以流式传输另一组额外统计数据，请选择 Add additional statistics（添加额外统计数据）。每个指标可以包含多达 20 个额外统计数据，一个指标流中有多达 100 个指标可以包含额外统计数据。

流式传输额外统计数据会产生更多费用。有关更多信息，请参阅 可以流式传输的统计数据。

有关额外统计数据的定义，请参阅 CloudWatch 统计数据定义。

10. （可选）可以在 Metric stream name（指标流名称）下自定义新指标流的名称。
11. 选择 Create metric filter（创建指标流）。

AWS CLI 或 AWS API

使用以下步骤创建 CloudWatch 指标流。

使用 AWS CLI 或 AWS API 创建指标流

1. 如果您要流式传输到 Amazon S3，请先创建存储桶。有关更多信息，请参阅创建存储桶。
2. 创建 Firehose 传输流。有关更多信息，请参阅创建 Firehose 流。
3. 创建使 CloudWatch 能够向 Firehose 传输流写入的 IAM 角色。有关该角色内容的更多信息，请参阅 CloudWatch 和 Firehose 之间的信任关系。
4. 使用 `aws cloudwatch put-metric-stream` CLI 命令或 `PutMetricStream` API 来创建 CloudWatch 指标流。

AWS CloudFormation

您可以使用 AWS CloudFormation 设置指标流。有关更多信息，请参阅 AWS::CloudWatch::MetricStream。

使用 AWS CloudFormation 创建指标流

1. 如果您要流式传输到 Amazon S3，请先创建存储桶。有关更多信息，请参阅创建存储桶。
2. 创建 Firehose 传输流。有关更多信息，请参阅创建 Firehose 流。
3. 创建使 CloudWatch 能够向 Firehose 传输流写入的 IAM 角色。有关该角色内容的更多信息，请参阅 CloudWatch 和 Firehose 之间的信任关系。

4. 在 AWS CloudFormation 中创建指标流。有关更多信息，请参阅 AWS::CloudWatch::MetricStream。

AWS Cloud Development Kit (AWS CDK)

您可以使用 AWS Cloud Development Kit (AWS CDK) 设置指标流。

使用 AWS CDK 创建指标流

1. 如果您要流式传输到 Amazon S3，请先创建存储桶。有关更多信息，请参阅创建存储桶。
2. 创建 Firehose 传输流。有关更多信息，请参阅创建 Amazon Data Firehose 传输流。
3. 创建使 CloudWatch 能够向 Firehose 传输流写入的 IAM 角色。有关该角色内容的更多信息，请参阅 CloudWatch 和 Firehose 之间的信任关系。
4. 创建指标流。指标流资源在 AWS CDK 中作为名为 `CfnMetricStream` 的 Level 1 (L1) Construct 提供。有关更多信息，请参阅使用 L1 Construct。

使用 Amazon S3 快速设置

如果您想快速设置传输到 Amazon S3 的流，并且除了支持的 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0 格式之外，您不需要任何格式转换，S3 快速设置方法非常适合。CloudWatch 将创建所有必要的资源，包括 Firehose 传输流和必要的 IAM 角色。此选项的默认格式为 JSON，但您可以在设置流时更改格式。

或者，如果您希望最终格式为 Parquet 格式或优化的行列式（ORC），则应改为按照 使用 Firehose 自定义设置 中的步骤进行操作。

CloudWatch 控制台

本部分介绍如何通过 CloudWatch 控制台使用“S3 快速设置”设置指标流 Amazon S3。

使用“S3 快速设置”设置指标流

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标）、Streams（流）。然后选择 Create metric stream（创建指标流）。
3. （可选）如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，您可以选择是否包含来自此指标流中关联源账户的指标。要包含来自源账户的指标，请选择 Include source account metrics（包含源账户指标）。

4. 选择S3 快速设置。CloudWatch 将创建所有必要的资源，包括 Firehose 传输流和必要的 IAM 角色。此选项的默认格式为 JSON，但您可以在此程序的稍后步骤中更改格式。
5. （可选）选择选择现有资源以使用现有 S3 存储桶或现有 IAM 角色，而不是让 CloudWatch 为您创建新的存储桶或 IAM 角色。
6. （可选）要更改方案的默认输出格式，请选择 Change output format（更改输出格式）。支持的格式为 JSON、OpenTelemetry 1.0.0 和 OpenTelemetry 0.7.0。
7. 对于要进行流式传输的指标，请选择所有指标或选择指标。

   如果您选择所有指标，则该账户的所有指标都将包含在流中。

   请仔细考虑是否要流式传输所有指标，因为流式传输的指标越多，您的指标流费用就越高。

   如果选择的是选择指标，请执行以下操作中的一项：

   - 要流式传输大多数指标命名空间，请选择排除，然后选择要排除的命名空间或指标。在排除中指定命名空间时，可以选择该命名空间中要排除的一些特定指标。如果选择排除某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将被排除。
   - 要在指标流中仅包含少数指标命名空间或指标，请选择包含，然后选择要包含的命名空间或指标。如果选择包含某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将包含在内。
8. （可选）要流式传输其中除 Minimum、Maximum、SampleCount 和 Sum 外的一些指标的其他统计数据，请选择添加其他统计数据。要么选择 Add recommended metrics（添加推荐的指标）添加一些常用的统计数据，要么手动选择要针对其流式传输额外统计数据的命名空间和指标名称。接下来，选择要流式传输的额外统计数据。

   然后，要选择另一组指标，以流式传输另一组额外统计数据，请选择 Add additional statistics（添加额外统计数据）。每个指标可以包含多达 20 个额外统计数据，一个指标流中有多达 100 个指标可以包含额外统计数据。

   流式传输额外统计数据会产生更多费用。有关更多信息，请参阅 可以流式传输的统计数据。

   有关额外统计数据的定义，请参阅 CloudWatch 统计数据定义。
9. （可选）可以在 Metric stream name（指标流名称）下自定义新指标流的名称。
10. 选择 Create metric filter（创建指标流）。

合作伙伴快速设置

CloudWatch 为以下第三方合作伙伴提供快速设置功能体验。要使用此工作流，您只需提供目标 URL 和目标的 API 密钥。CloudWatch 将负责其余的设置，包括创建 Firehose 传输流和所需的 IAM 角色。

> **Important**
>
> 在使用合作伙伴快速设置创建指标流之前，我们强烈建议您阅读以下列表中链接的该合作伙伴的文档。

- Datadog
- Dynatrace
- New Relic
- Splunk Observability Cloud
- SumoLogic

当您设置要传输到其中一个合作伙伴的指标流时，将使用一些默认设置创建该流，如以下各部分所列。

主题

- 使用“合作伙伴快速设置”设置指标流
- Datadog 流默认设置
- Dynatrace 流默认设置
- New Relic 流默认设置
- Splunk Observability Cloud 流默认设置
- Sumo Logic 流默认设置

使用“合作伙伴快速设置”设置指标流

CloudWatch 为某些第三方合作伙伴提供了快速设置功能选项。在开始执行本部分中的步骤之前，您必须先了解合作伙伴的某些信息。这些信息可能包括目标 URL 和/或合作伙伴目标的 API 密钥。您还应该阅读上一部分中链接的合作伙伴网站上的文档，以及以下各部分中列出的针对相应合作伙伴的默认设置。

要流式传输到快速设置功能不支持的第三方目标，您可以按照 使用 Firehose 自定义设置 中的说明，使用 Firehose 设置流，然后将这些指标从 Firehose 发送到最终目标。

使用“合作伙伴快速设置”创建传输到第三方提供商的指标流

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标）、Streams（流）。然后选择 Create metric stream（创建指标流）。
3. （可选）如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，您可以选择是否包含来自此指标流中关联源账户的指标。要包含来自源账户的指标，请选择 Include source account metrics（包含源账户指标）。
4. 选择快速设置 Amazon Web Services 合作伙伴
5. 选择要将指标流式传输到的合作伙伴的名称。
6. 对于端点 URL，输入目标 URL。
7. 对于访问密钥或 API 密钥，输入相应合作伙伴的访问密钥。并非所有合作伙伴都需要访问密钥。
8. 对于要进行流式传输的指标，请选择所有指标或选择指标。

   如果您选择所有指标，则该账户的所有指标都将包含在流中。

   请仔细考虑是否要流式传输所有指标，因为流式传输的指标越多，您的指标流费用就越高。

   如果选择的是选择指标，请执行以下操作中的一项：

   - 要流式传输大多数指标命名空间，请选择排除，然后选择要排除的命名空间或指标。在排除中指定命名空间时，可以选择该命名空间中要排除的一些特定指标。如果选择排除某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将被排除。
   - 要在指标流中仅包含少数指标命名空间或指标，请选择包含，然后选择要包含的命名空间或指标。如果选择包含某个命名空间，但随后没有选择该命名空间中的指标，则该命名空间中的所有指标都将包含在内。
9. （可选）要流式传输其中除 Minimum、Maximum、SampleCount 和 Sum 外的一些指标的其他统计数据，请选择添加其他统计数据。要么选择 Add recommended metrics（添加推荐的指标）添加一些常用的统计数据，要么手动选择要针对其流式传输额外统计数据的命名空间和指标名称。接下来，选择要流式传输的额外统计数据。

   然后，要选择另一组指标，以流式传输另一组额外统计数据，请选择 Add additional statistics（添加额外统计数据）。每个指标可以包含多达 20 个额外统计数据，一个指标流中有多达 100 个指标可以包含额外统计数据。

   流式传输额外统计数据会产生更多费用。有关更多信息，请参阅 可以流式传输的统计数据。

   有关额外统计数据的定义，请参阅 CloudWatch 统计数据定义。

10. （可选）可以在 Metric stream name（指标流名称）下自定义新指标流的名称。
11. 选择 Create metric filter（创建指标流）。

Datadog 流默认设置

传输到 Datadog 的合作伙伴快速设置流使用以下默认设置：

- 输出格式：OpenTelemetry 0.7.0
- Firehose 流内容编码 GZIP
- Firehose 流缓冲选项 间隔为 60 秒，大小为 4MB
- Firehose 流重试选项 持续时间为 60 秒

当您使用“合作伙伴快速设置”创建传输到 Datadog 的指标流并流式传输某些指标时，默认情况下，这些指标会包含一些其他统计数据。流式传输其他统计数据可能会产生额外费用。有关这些统计数据及其费用的更多信息，请参阅 可以流式传输的统计数据。

如果您选择流式传输默认会流式传输其他统计数据的指标，以下列表将显示这些指标。在启动流式传输之前，您可以选择取消选择这些其他统计数据。

- **AWS/Lambda** 中的 **Duration**：p50、p80、p95、p99、p99.9
- **AWS/Lambda** 中的 **PostRuntimeExtensionDuration**：p50、p99
- **AWS/S3** 中的 **FirstByteLatency** 和 **TotalRequestLatency**：p50、p90、p95、p99、p99.9
- AWS/ApplicationELB 中 **AWS/Polly** 和 **TargetResponseTime** 中的 **ResponseLatency**：p50、p90、p95、p99
- **AWS/ApiGateway** 中的 **Latency** 和 **IntegrationLatency**：p90、p95、p99
- **AWS/ELB** 中的 **Latency** 和 **TargetResponseTime**：p95、p99
- **AWS/AppRunner** 中的 **RequestLatency**：p50、p95、p99
- **AWS/States** 中的 **ActivityTime**、**ExecutionTime**、**LambdaFunctionRunTime**、**LambdaFunctionScheduleTime**、和 **ActivityScheduleTime**：p95、p99
- **AWS/MediaLive** 中的 **EncoderBitRate**、**ConfiguredBitRate** 和 **ConfiguredBitRateAvailable**：p90
- **AWS/AppSync** 中的 **Latency**：p90

Dynatrace 流默认设置

传输到 Dynatrace 的合作伙伴快速设置流使用以下默认设置：

- 输出格式：OpenTelemetry 0.7.0
- Firehose 流内容编码 GZIP
- Firehose 流缓冲选项 间隔为 60 秒，大小为 5MB
- Firehose 流重试选项 持续时间为 600 秒

New Relic 流默认设置

传输到 New Relic 的合作伙伴快速设置流使用以下默认设置：

- 输出格式：OpenTelemetry 0.7.0
- Firehose 流内容编码 GZIP
- Firehose 流缓冲选项 间隔为 60 秒，大小为 1MB
- Firehose 流重试选项 持续时间为 60 秒

Splunk Observability Cloud 流默认设置

传输到 Splunk Observability Cloud 的合作伙伴快速设置流使用以下默认设置：

- 输出格式：OpenTelemetry 0.7.0
- Firehose 流内容编码 GZIP
- Firehose 流缓冲选项 间隔为 60 秒，大小为 1MB
- Firehose 流重试选项 持续时间为 300 秒

Sumo Logic 流默认设置

传输到 Sumo Logic 的合作伙伴快速设置流使用以下默认设置：

- 输出格式：OpenTelemetry 0.7.0
- Firehose 流内容编码 GZIP
- Firehose 流缓冲选项 间隔为 60 秒，大小为 1MB
- Firehose 流重试选项 持续时间为 60 秒

可以流式传输的统计数据

指标流始终包括以下统计数据：Minimum、Maximum、SampleCount 和 Sum。您还可以选择在指标流中包括以下其他统计数据。此选择基于每个指标。有关这些统计数据的更多信息，请参阅 CloudWatch 统计数据定义。

- 百分位数值，例如 p95 或 p99（适用于 JSON 或 OpenTelemetry 格式的流）
- 切尾均值（仅适用于 JSON 格式的流）
- 缩尾均值（仅适用于 JSON 格式的流）
- 切尾计数（仅适用于 JSON 格式的流）
- 切尾总和（仅适用于 JSON 格式的流）
- 百分位数排名（仅适用于 JSON 格式的流）
- 四分位数均值（仅适用于 JSON 格式的流）

流式传输额外统计数据会产生额外费用。流式传输某一特定指标的其中 1 个到 5 个额外统计数据，将作为一次额外指标更新来计费。此后，每增加一组（最多包含其中 5 个统计数据），将作为另一次指标更新来计费。

例如，假设对于某一指标，您正在流式传输以下 6 个额外统计数据：p95、p99、p99.9、切尾均值、缩尾均值和切尾总和。此指标的每次更新均作为三次指标更新计费：一个用于包含默认统计数据的指标更新，一个用于前 5 个额外统计数据，还有一个用于第 6 个额外统计数据。再添加最多 4 个额外统计数据，共 10 个，不会增加计费，但是第 11 个额外统计数据就会增加计费。

当您指定指标名称和命名空间组合，以流式传输额外统计数据时，该指标名称和命名空间的所有维度组合都将与这些额外统计数据一起流式传输。

CloudWatch 指标流发布了一个新指标，TotalMetricUpdate，它反映了指标更新加上由流式传输额外统计数据产生的额外指标更新的基本数量。有关更多信息，请参阅 使用 CloudWatch 指标监控您的指标流。

有关更多信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> 有些指标不支持百分位数。这些指标的百分位数统计数据将从流式传输中排除，因而不会产生指标流费用。这些不支持百分位数的统计数据的一个示例是 AWS/ECS 命名空间的一些指标。

仅当您配置的额外统计数据与流的筛选条件相匹配时，才会流式传输这些统计数据。例如，如果您创建的流仅将 EC2 和 RDS 包含在包含筛选条件中，然后您的统计数据配置列出 EC2 和 Lambda，则该流将包含 EC2 指标（其中包含额外统计数据）、RDS 指标（仅包含默认统计数据），但根本不包含 Lambda 统计数据。

指标流操作和维护

指标流始终处于两种状态之一，即正在运行或已停止状态。

- 正在运行 – 指标流正在正确运行。由于指标流的筛选条件，可能没有任何已流式传输到目标位置的指标数据。
- 已停止 – 指标流已被某人明确停止，且不是因错误而停止。停止指标流以暂时暂停流式传输数据而不删除指标流可能会很有帮助。

如果停止并重新启动指标流，则在指标流停止期间发布到 CloudWatch 的指标数据不会回填到该指标流。

如果更改指标流的输出格式，在某些情况下，您可能会看到少量指标数据以新旧两种格式写入到目标位置。为了避免这种情况，您可以使用与当前传输流相同的配置创建新的 Firehose 传输流，然后更改为新的 Firehose 传输流并同时更改输出格式。这样，具有不同输出格式的 Kinesis 记录将存储在 Amazon S3 中的单独对象中。您稍后可以将流量导向回原始 Firehose 传输流，并删除另一个传输流。

查看、编辑、停止和启动指标流

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标）、Streams（流）。

   此时将显示流列表，并且 Status（状态）列显示每个流是处于正在运行还是已停止状态。
3. 要停止或启动指标流，请选择该流，然后选择 Stop（停止）或 Start（启动）。
4. 要查看有关指标流的详细信息，请选择该流，然后选择 View details（查看详细信息）。
5. 若要更改流的输出格式、筛选条件、目标 Firehose 流或角色，请选择编辑并执行所需的更改。

   如果您更改筛选条件，则在过渡期间指标数据中可能存在一些差异。

使用 CloudWatch 指标监控您的指标流

指标流会在 AWS/CloudWatch/MetricStreams 命名空间中发出关于自身运行状况和操作的 CloudWatch 指标。将发出以下指标。发出的这些指标都会存在具有 MetricStreamName 维度和

没有维度两种情况。您可以使用没有维度的指标来查看所有指标流的已聚合指标。您可以使用具有 MetricStreamName 维度的指标以查看仅与该指标流有关的指标。

对于所有这些指标，均仅针对处于 Running（正在运行）状态的指标流发出数值。

指标	描述
MetricUpdate	发送到指标流的指标更新次数。如果在某个时间段内没有流式传输指标更新，则在该时间段内不会发布此指标。 如果停止指标流，则将停止发出该指标，直到该指标流再次启动。 有效统计数据：Sum 单位：无
TotalMetricUpdate	计算方法为 MetricUpdate + 一个基于正在流式传输的额外统计数据的数字。 对于每个唯一的命名空间和指标名称组合，流式传输 1-5 个额外统计数据将向 TotalMetricUpdate 加 1，流式传输 6-10 个额外统计数据将向 TotalMetricUpdate 加 2，以此类推。 有效统计数据：Sum 单位：无
PublishErrorRate	在将数据放入 Firehose 传输流时发生的不可恢复的错误数。如果在某个时间段内没有发生错误，则在该时间段内不会发布此指标。 如果停止指标流，则将停止发出该指标，直到该指标流再次启动。 有效统计数据：Average，用于查看无法写入的指标更新速率。此值必须介于 0.0 到 1.0 之间。 单位：无

CloudWatch 和 Firehose 之间的信任关系

Firehose 传输流必须通过对 Firehose 具有写入权限的 IAM 角色来信任 CloudWatch。这些权限可仅限为 CloudWatch 指标流使用的单个 Firehose 传输流。IAM 角色必须信任 `streams.metrics.cloudwatch.amazonaws.com` 服务主体。

如果您使用 CloudWatch 控制台创建指标流，则可以让 CloudWatch 创建具有正确权限的角色。如果您使用其他方法创建指标流，或者希望创建 IAM 角色本身，则该角色必须包含以下权限策略和信任策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:firehose:region:account-id:deliverystream/*"
        }
    ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "streams.metrics.cloudwatch.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

指标数据由 CloudWatch 代表拥有该指标流资源的源将其流式传输到目标 Firehose 传输流。

指标流输出格式

CloudWatch 指标流中的数据可为 JSON 格式或 OpenTelemetry 格式。目前同时支持 OpenTelemetry 1.0.0 和 0.7.0 格式。

目录

- JSON 格式
  - 我应该对 JSON 输出格式使用哪个 AWS Glue 架构？
- OpenTelemetry 1.0.0 格式
  - 转换为 OpenTelemetry 1.0.0 格式
  - 如何解析 OpenTelemetry 1.0.0 消息
- OpenTelemetry 0.7.0 format
  - 转换为 OpenTelemetry 0.7.0 格式
  - 如何解析 OpenTelemetry 0.7.0 消息

JSON 格式

在使用 JSON 格式的 CloudWatch 指标流中，每条 Firehose 记录都包含多个由换行符（\n）分隔的 JSON 对象。每个对象包含单个指标的单一数据点。

所使用的 JSON 格式与 AWS Glue 和 Amazon Athena 完全兼容。如果您的 Firehose 传输流和 AWS Glue 表格的格式均正确，则格式可以自动转换为 Parquet 格式或优化的行列式（ORC）格式，然后再存储在 S3 中。有关转换格式的更多信息，请参阅在 Firehose 中转换输入记录格式。有关 AWS Glue 的正确格式的更多信息，请参阅 我应该对 JSON 输出格式使用哪个 AWS Glue 架构？。

在 JSON 格式中，`unit` 的有效值与 `MetricDatum` API 结构中 `unit` 的值相同。有关更多信息，请参阅 MetricDatum。`timestamp` 字段的值以纪元毫秒为单位，例如 `1616004674229`。

以下为格式示例。在本示例中，JSON 所用格式的目的是便于查看，但实际上，整个格式应位于一行上。

```
{
    "metric_stream_name": "MyMetricStream",
    "account_id": "1234567890",
    "region": "us-east-1",
    "namespace": "AWS/EC2",
    "metric_name": "DiskWriteOps",
    "dimensions": {
```

```
        "InstanceId": "i-123456789012"
    },
    "timestamp": 1611929698000,
    "value": {
        "count": 3.0,
        "sum": 20.0,
        "max": 18.0,
        "min": 0.0,
        "p99": 17.56,
        "p99.9": 17.8764,
        "TM(25%:75%)": 16.43
    },
    "unit": "Seconds"
}
```

我应该对 JSON 输出格式使用哪个 AWS Glue 架构？

以下是 AWS Glue 表的 `StorageDescriptor` 的 JSON 表示（之后由 Firehose 使用）示例。有关 `StorageDescriptor` 的更多信息，请参阅 StorageDescriptor。

```
{
  "Columns": [
    {
      "Name": "metric_stream_name",
      "Type": "string"
    },
    {
      "Name": "account_id",
      "Type": "string"
    },
    {
      "Name": "region",
      "Type": "string"
    },
    {
      "Name": "namespace",
      "Type": "string"
    },
    {
      "Name": "metric_name",
      "Type": "string"
    },
    {
```

```
      "Name": "timestamp",
      "Type": "timestamp"
    },
    {
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
    {
      "Name": "value",
      "Type":
 "struct<min:double,max:double,count:double,sum:double,p99:double,p99.9:double>"
    },
    {
      "Name": "unit",
      "Type": "string"
    }
  ],
  "Location": "s3://my-s3-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}
```

上面的示例适用于以 JSON 格式在 Amazon S3 上写入的数据。将以下字段中的值替换为指定的值，以便以 Parquet 格式或 Optimized Row Columnar (ORC) 格式存储数据。

- Parquet：
  - inputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
  - parameters.classification: parquet
- ORC：
  - inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde

- parameters.classification: orc

OpenTelemetry 1.0.0 格式

> **Note**
>
> 如果选择 OpenTelemetry 1.0.0 格式，指标属性会被编码为一个 `KeyValue` 对象列表，而不是 0.7.0 格式中使用的 `StringKeyValue` 类型。作为消费端，这是 0.7.0 和 1.0.0 格式之间的唯一重大变化。从 0.7.0 原型文件生成的解析器无法解析以 1.0.0 格式编码的指标属性。反之亦然，从 1.0.0 原型文件生成的解析器无法解析以 0.7.0 格式编码的指标属性。

OpenTelemetry 是工具、API 和软件开发工具包的集合。您可以使用它来测量、生成、收集及导出遥测数据（指标、日志和跟踪）以进行分析。OpenTelemetry 属于云原生计算基金会。有关更多信息，请参阅 OpenTelemetry。

有关完整 OpenTelemetry 1.0.0 规范的信息，请参阅 Release version 1.0.0。

一条 Kinesis 记录可以包含一个或多个 `ExportMetricsServiceRequest` OpenTelemetry 数据结构。每个数据结构都以一个带有指示记录长度（字节）的 `UnsignedVarInt32` 的标头开头。每个 `ExportMetricsServiceRequest` 可同时包含来自多个指标的数据。

以下是 `ExportMetricsServiceRequest` OpenTelemetry 数据结构形式的消息的字符串表示。OpenTelemetry 会序列化 Google Protocol Buffers 二进制协议，此协议人类不可读。

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "123456789012"
      }
    }
    attributes {
```

```
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  scope_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "NoneTranslated"
      summary {
        data_points {
          start_time_unix_nano: 60000000000
          time_unix_nano: 120000000000
          count: 1
          sum: 1.0
          quantile_values {
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
            }
          }
```

```
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
            }
          }
          attributes {
            key: "Dimensions"
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
                  }
                }
              }
            }
          }
        }
        data_points {
          start_time_unix_nano: 70000000000
          time_unix_nano: 130000000000
          count: 2
          sum: 5.0
          quantile_values {
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
            }
          }
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
            }
          }
```

```
          attributes {
            key: "Dimensions"
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

用于序列化 OpenTelemetry 指标数据的顶级对象

`ExportMetricsServiceRequest` 是用于序列化 OpenTelemetry 导出器负载的顶级包装器。它包含一个或多个 `ResourceMetrics`。

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` 是表示 MetricData 对象的顶级对象。

```
// A collection of ScopeMetrics from a Resource.
message ResourceMetrics {
  reserved 1000;

  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;
```

```
  // A list of metrics that originate from a resource.
  repeated ScopeMetrics scope_metrics = 2;

  // This schema_url applies to the data in the "resource" field. It does not apply
  // to the data in the "scope_metrics" field which have their own schema_url field.
  string schema_url = 3;
}
```

资源对象

Resource 对象是一个值对对象，其中包含有关生成指标的资源的一些信息。对于由 AWS 创建的指标，数据结构包含与指标相关的资源的 Amazon Resource Name (ARN)，例如 EC2 实例或 S3 存储桶。

Resource 对象包含名为 attributes 的属性，其会存储键值对列表。

- cloud.account.id 包含账户 ID
- cloud.region 包含区域
- aws.exporter.arn 包含指标流 ARN
- cloud.provider 始终为 aws。

```
// Resource information.
message Resource {
  // Set of attributes that describe the resource.
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
 then
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

ScopeMetrics 对象

scope 字段将不会填充。我们仅填充正在导出的指标字段。

```
// A collection of Metrics produced by an Scope.
```

```
message ScopeMetrics {
  // The instrumentation scope information for the metrics in this message.
  // Semantically when InstrumentationScope isn't set, it is equivalent with
  // an empty instrumentation scope name (unknown).
  opentelemetry.proto.common.v1.InstrumentationScope scope = 1;

  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;

  // This schema_url applies to all metrics in the "metrics" field.
  string schema_url = 3;
}
```

指标对象

指标对象包含一些元数据和一个 Summary 数据字段，该字段包含一个 SummaryDataPoint 列表。

对于指标流，元数据如下所示：

- name 将为 amazonaws.com/*metric_namespace*/*metric_name*
- description 将为空
- unit 将通过将指标基准单位映射为计量单位统一代码的变体（区分大小写）来填充。有关更多信息，请参阅 转换为 OpenTelemetry 1.0.0 格式 和计量单位统一代码。
- type 将为 SUMMARY

```
message Metric {
  reserved 4, 6, 8;

  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  // Data determines the aggregation type (if any) of the metric, what is the
  // reported value type for the data points, as well as the relatationship to
  // the time interval over which they are reported.
```

```
  oneof data {
    Gauge gauge = 5;
    Sum sum = 7;
    Histogram histogram = 9;
    ExponentialHistogram exponential_histogram = 10;
    Summary summary = 11;
  }
}

message Summary {
  repeated SummaryDataPoint data_points = 1;
}
```

SummaryDataPoint 对象

SummaryDataPoint 对象包含 DoubleSummary 指标时间序列中单个数据点的值。

```
// SummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message SummaryDataPoint {
  reserved 1;

  // The set of key/value pairs that uniquely identify the timeseries from
  // where this point belongs. The list may be empty (may contain 0 elements).
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 7;

  // StartTimeUnixNano is optional but strongly encouraged, see the
  // the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 start_time_unix_nano = 2;

  // TimeUnixNano is required, see the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;
```

```
  // sum of the values in the population. If count is zero then this field
  // must be zero.
  //
  // Note: Sum should only be filled out when measuring non-negative discrete
  // events, and is assumed to be monotonic over the values of these events.
  // Negative events *can* be recorded, but sum should not be filled out when
  // doing so.  This is specifically to enforce compatibility w/ OpenMetrics,
  // see: https://github.com/OpenObservability/OpenMetrics/blob/main/specification/
OpenMetrics.md#summary
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  //
  // See the following issue for more context:
  // https://github.com/open-telemetry/opentelemetry-proto/issues/125
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    //
    // Quantile values must NOT be negative.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;

  // Flags that apply to this specific data point.  See DataPointFlags
  // for the available flags and their meaning.
  uint32 flags = 8;
}
```

有关更多信息，请参阅 [转换为 OpenTelemetry 1.0.0 格式](#)。

转换为 OpenTelemetry 1.0.0 格式

CloudWatch 会执行一些转换，将 CloudWatch 数据转换为 OpenTelemetry 格式。

转换命名空间、指标名称和维度

这些属性是在映射中编码的键值对。

- 一个属性的键是 `Namespace`，其值是指标的命名空间
- 一个属性的键是 `MetricName`，其值是指标的名称
- 一个键值对的键是 `Dimensions`，其值是一个键值对嵌套列表。此列表中的每个键值对都映射到一个 CloudWatch 指标维度，其中键值对的键是维度的名称，其值是维度的值。

转换平均值、总和、样本计数、最小值和最大值

摘要数据点使 CloudWatch 能够使用一个数据点导出所有这些统计数据。

- `startTimeUnixNano` 包含 CloudWatch `startTime`
- `timeUnixNano` 包含 CloudWatch `endTime`
- `sum` 包含总和统计数据。
- `count` 包含样本数统计数据。
- `quantile_values` 包含两个 `valueAtQuantile.value` 对象：
  - `valueAtQuantile.quantile = 0.0` 与 `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` 与 `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` 与 `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` 与 `valueAtQuantile.value = Max value`

使用该指标流的资源可以按总和/样本数来计算平均值统计数据。

转换单位

CloudWatch 单位映射到计量单位统一代码区分大小写的变体，如下表所示。有关更多信息，请参阅计量单位统一代码。

CloudWatch	OpenTelemetry
秒	s
秒	s
微秒	us

CloudWatch	OpenTelemetry
毫秒	ms
字节	By
千字节	kBy
兆字节	MBy
千兆字节	GBy
千吉字节	TBy
Bits	bit
千位	kbit
兆位	MBit
千兆位	GBit
太位	Tbit
百分比	%
计数	{Count}
无	1

由斜线组合而成的单位通过同时对斜线前后两个单位应用 OpenTelemetry 转换来进行映射。例如，字节/秒映射为 By/s。

如何解析 OpenTelemetry 1.0.0 消息

本节提供了有助于您开始解析 OpenTelemetry 1.0.0 的信息。

首先，您应获得特定于语言的绑定，这样您才能够以首选语言解析 OpenTelemetry 1.0.0 消息。

获取特定于语言的绑定

- 根据您的首选语言选择以下步骤。

- 若要使用 Java，请将以下 Maven 依赖项添加到您的 Java 项目中：OpenTelemetry Java >> 0.14.1。
- 若要使用任何其他语言，请按照下列步骤操作：

  a. 检查生成类中的列表，确保您的语言受支持。

  b. 按照下载协议缓冲区中的步骤安装 Protobuf 编译器。

  c. 在 Release version 1.0.0 中下载 OpenTelemetry 0.7.0 ProtoBuf 定义。

  d. 确认您位于下载的 OpenTelemetry 0.7.0 ProtoBuf 定义的根文件夹中。创建一个 `src` 文件夹，然后运行命令以生成特定于语言的绑定。有关更多信息，请参阅生成类。

     以下示例展示了如何生成 Javascript 绑定。

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

以下部分包括使用特定于语言的绑定的示例，您可以使用前面的说明来构建这些绑定。

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
```

```
            UnsignedVarInt32 indicating the record length in bytes:
             ------ -------------------------- ------ ----------------------
            |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
             ------ -------------------------- ------ ----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

本示例假定具有所生成的绑定的根文件夹为 `./`

函数 `parseRecord` 的数据参数可以是以下类型之一：

- `Uint8Array`，此类型最佳
- `Buffer`，在节点下最佳
- `Array.number`，8 位整数

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        const reader = new pb.BinaryReader(data)
```

```
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

您必须自行读取 var-int 分隔符或使用内部方法 _VarintBytes(size) 和 _DecodeVarint32(buffer, position)。它们会返回刚好位于缓冲区中大小字节之后的位置。读取端构建一个新的缓冲区，该缓冲区仅限于读取消息的字节。

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

使用 Buffer.DecodeMessage()。

C#

使用 CodedInputStream。此类可以读取分隔了大小的消息。

C++

`google/protobuf/util/delimited_message_util.h` 中描述的函数可以读取分隔了大小的消息。

其他语言

有关使用其他语言，请参阅下载协议缓冲区。

在实现解析器时，请注意，一条 Kinesis 记录可以包含多个 `ExportMetricsServiceRequest` 协议缓冲区消息，每个消息都以具有指示记录长度（字节）的 `UnsignedVarInt32` 的标头开头。

OpenTelemetry 0.7.0 format

OpenTelemetry 是工具、API 和软件开发工具包的集合。您可以使用它来测量、生成、收集及导出遥测数据（指标、日志和跟踪）以进行分析。OpenTelemetry 属于云原生计算基金会。有关更多信息，请参阅 OpenTelemetry。

有关完整 OpenTelemetry 0.7.0 规范的信息，请参阅 v0.7.0 版本。

一条 Kinesis 记录可以包含一个或多个 `ExportMetricsServiceRequest` OpenTelemetry 数据结构。每个数据结构都以一个带有指示记录长度（字节）的 `UnsignedVarInt32` 的标头开头。每个 `ExportMetricsServiceRequest` 可同时包含来自多个指标的数据。

以下是 `ExportMetricsServiceRequest` OpenTelemetry 数据结构形式的消息的字符串表示。OpenTelemetry 会序列化 Google Protocol Buffers 二进制协议，此协议人类不可读。

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
```

```
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
      double_summary {
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 1604948400000000000
          time_unix_nano: 1604948460000000000
          count: 1
          sum: 1.0
          quantile_values {
            quantile: 0.0
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
```

```
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
        }
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 1604948460000000000
          time_unix_nano: 1604948520000000000
          count: 2
          sum: 5.0
          quantile_values {
            quantile: 0.0
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
        }
      }
    }
  }
}
```

用于序列化 OpenTelemetry 指标数据的顶级对象

`ExportMetricsServiceRequest` 是用于序列化 OpenTelemetry 导出器负载的顶级包装器。它包含一个或多个 `ResourceMetrics`。

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

ResourceMetrics 是表示 MetricData 对象的顶级对象。

```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
}
```

资源对象

Resource 对象是一个值对对象，其中包含有关生成指标的资源的一些信息。对于由 AWS 创建的指标，数据结构包含与指标相关的资源的 Amazon Resource Name (ARN)，例如 EC2 实例或 S3 存储桶。

Resource 对象包含名为 attributes 的属性，其会存储键值对列表。

- cloud.account.id 包含账户 ID
- cloud.region 包含区域
- aws.exporter.arn 包含指标流 ARN
- cloud.provider 始终为 aws。

```
// Resource information.
message Resource {
  // Set of labels that describe the resource.
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;
```

```
  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

InstrumentationLibraryMetrics 对象

将不会填写 instrumentation_library 字段。我们将仅填写正在导出的指标字段。

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
  // The instrumentation library information for the metrics in this message.
  // If this field is not set then no library info is known.
  opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;
}
```

指标对象

指标对象包含一个 DoubleSummary 数据字段，该字段包含一个 DoubleSummaryDataPoint 列表。

```
message Metric {
  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  oneof data {
    IntGauge int_gauge = 4;
    DoubleGauge double_gauge = 5;
    IntSum int_sum = 6;
    DoubleSum double_sum = 7;
    IntHistogram int_histogram = 8;
    DoubleHistogram double_histogram = 9;
    DoubleSummary double_summary = 11;
  }
```

```
}

message DoubleSummary {
  repeated DoubleSummaryDataPoint data_points = 1;
}
```

MetricDescriptor 对象

MetricDescriptor 对象包含元数据。有关更多信息，请参阅 GitHub 上的 metrics.proto。

对于指标流，MetricDescriptor 包含以下内容：

- name 将为 amazonaws.com/*metric_namespace*/*metric_name*
- description 将为空。
- unit 将通过将指标基准单位映射为计量单位统一代码的变体（区分大小写）来填充。有关更多信息，请参阅 转换为 OpenTelemetry 0.7.0 格式 和计量单位统一代码。
- type 将为 SUMMARY。

DoubleSummaryDataPoint 对象

DoubleSummaryDataPoint 对象包含 DoubleSummary 指标时间序列中单个数据点的值。

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
  // The set of labels that uniquely identify this timeseries.
  repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

  // start_time_unix_nano is the last time when the aggregation value was reset
  // to "zero". For some metric types this is ignored, see data types for more
  // details.
  //
  // The aggregation value is over the time interval (start_time_unix_nano,
  // time_unix_nano].
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  //
  // Value of 0 indicates that the timestamp is unspecified. In that case the
  // timestamp may be decided by the backend.
  fixed64 start_time_unix_nano = 2;
```

```
  // time_unix_nano is the moment when this aggregation value was reported.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;
}
```

有关更多信息，请参阅 转换为 OpenTelemetry 0.7.0 格式。

转换为 OpenTelemetry 0.7.0 格式

CloudWatch 会执行一些转换，将 CloudWatch 数据转换为 OpenTelemetry 格式。

转换命名空间、指标名称和维度

这些属性是在映射中编码的键值对。

- 其中一对包含指标的命名空间

- 另一对包含指标的名称
- 对于每个维度，CloudWatch 都存储以下键值对：`metricDatum.Dimensions[i].Name`, `metricDatum.Dimensions[i].Value`

转换平均值、总和、样本计数、最小值和最大值

摘要数据点使 CloudWatch 能够使用一个数据点导出所有这些统计数据。

- `startTimeUnixNano` 包含 CloudWatch `startTime`
- `timeUnixNano` 包含 CloudWatch `endTime`
- `sum` 包含总和统计数据。
- `count` 包含样本数统计数据。
- `quantile_values` 包含两个 `valueAtQuantile.value` 对象：
  - `valueAtQuantile.quantile = 0.0` 与 `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` 与 `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` 与 `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` 与 `valueAtQuantile.value = Max value`

使用该指标流的资源可以按总和/样本数来计算平均值统计数据。

转换单位

CloudWatch 单位映射到计量单位统一代码区分大小写的变体，如下表所示。有关更多信息，请参阅计量单位统一代码。

CloudWatch	OpenTelemetry
秒	s
秒	s
Microsecond	us
毫秒	ms
字节	By

CloudWatch	OpenTelemetry
千字节	kBy
兆字节	MBy
千兆字节	GBy
千吉字节	TBy
Bits	bit
千位	kbit
兆位	MBit
千兆位	GBit
太位	Tbit
百分比	%
计数	{Count}
无	1

由斜线组合而成的单位通过同时对斜线前后两个单位应用 OpenTelemetry 转换来进行映射。例如，字节/秒映射为 By/s。

如何解析 OpenTelemetry 0.7.0 消息

本部分提供了有助于您开始解析 OpenTelemetry 0.7.0 的信息。

首先，您应获得特定于语言的绑定，这样您能够以您的首选语言解析 OpenTelemetry 0.7.0 消息。

获取特定于语言的绑定

- 根据您的首选语言选择以下步骤。
  - 若要使用 Java，请将以下 Maven 依赖项添加到您的 Java 项目中：OpenTelemetry Java >> 0.14.1。
  - 若要使用任何其他语言，请按照下列步骤操作：

a. 检查生成类中的列表，确保您的语言受支持。

b. 按照下载协议缓冲区中的步骤安装 Protobuf 编译器。

c. 在 v0.7.0 版本中下载 OpenTelemetry 0.7.0 ProtoBuf 定义。

d. 确认您位于下载的 OpenTelemetry 0.7.0 ProtoBuf 定义的根文件夹中。创建一个 `src` 文件夹，然后运行命令以生成特定于语言的绑定。有关更多信息，请参阅生成类。

   以下示例展示了如何生成 Javascript 绑定。

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

以下部分包括使用特定于语言的绑定的示例，您可以使用前面的说明来构建这些绑定。

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ --------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ --------------------------- ------ ----------------------
```

```
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

本示例假定具有所生成的绑定的根文件夹为 `./`

函数 `parseRecord` 的数据参数可以是以下类型之一：

- `Uint8Array`，此类型最佳
- `Buffer`，在节点下最佳
- `Array.number`，8 位整数

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength
```

```
        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

您必须自行读取 var-int 分隔符或使用内部方法 _VarintBytes(size) 和 _DecodeVarint32(buffer, position)。它们会返回刚好位于缓冲区中大小字节之后的位置。读取端构建一个新的缓冲区，该缓冲区仅限于读取消息的字节。

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

使用 Buffer.DecodeMessage()。

C#

使用 CodedInputStream。此类可以读取分隔了大小的消息。

C++

google/protobuf/util/delimited_message_util.h 中描述的函数可以读取分隔了大小的消息。

其他语言

有关使用其他语言，请参阅下载协议缓冲区。

在实现解析器时，请注意，一条 Kinesis 记录可以包含多个 `ExportMetricsServiceRequest` 协议缓冲区消息，每个消息都以具有指示记录长度（字节）的 `UnsignedVarInt32` 的标头开头。

故障排除

如果您在最终目标位置看不到指标数据，请检查以下各项：

- 检查指标流是否处于正在运行状态。有关如何使用 CloudWatch 控制台执行此操作的步骤，请参阅指标流操作和维护。
- 无法流式传输发布时间超过两天的指标。要确定是否流式传输特定指标，请在 CloudWatch 控制台中绘制指标图表，然后检查最后一个可见数据点的存在时间。如果发布时间超过两天，那么指标流将不会获取此指标。
- 检查指标流发出的指标。在 CloudWatch 控制台中，在 Metrics（指标）下，查看 MetricUpdate、TotalMetricUpdate 和 PublishErrorRate 指标的 AWS/CloudWatch/MetricStreams 命名空间。
- 如果 PublishErrorRate 指标较高，请确认 Firehose 传输流使用的目标存在，并且指标流配置中指定的 IAM 角色授予了 `CloudWatch` 服务主要权限以写入其中。有关更多信息，请参阅 CloudWatch 和 Firehose 之间的信任关系。
- 检查 Firehose 传输流是否有权限写入最终目标位置。
- 在 Firehose 控制台中，查看用于指标流的 Firehose 传输流，并检查监控选项卡以查看 Firehose 传输流是否在接收数据。
- 确认您已为 Firehose 传输流配置了正确的详细信息。
- 检查 Firehose 传输流写入的最终目标位置的任何可用日志或指标。
- 若要获取更多详细信息，请在 Firehose 传输流上启用 CloudWatch Logs 错误日志记录。有关更多信息，请参阅使用 CloudWatch Logs 监控 Amazon Data Firehose。

> **Note**
>
> 在发送特定指标和时间戳的数据点后，即使其值稍后更改，也不会再次发送该数据点。

查看可用的指标

指标首先按命名空间进行分组，然后按各命名空间内的各种维度组合进行分组。例如，您可以查看所有 EC2 指标、按实例分组的 EC2 指标或按 Auto Scaling 组分组的 EC2 指标。

只有您使用的 AWS 服务会将指标发送到 Amazon CloudWatch。

有关向 CloudWatch 发送指标的 AWS 服务列表，请参阅 发布 CloudWatch 指标的 AWS 服务。在此页面中，您还可以查看这些服务中的每个服务所发布的指标和维度。

> **Note**
>
> 控制台中不会显示在过去两周内没有任何新数据点的指标。当您在控制台的 All metrics（全部指标）选项卡的搜索框中键入指标名称或维度名称时，它们也不会显示，并且 list-metrics 命令的结果中不会返回它们。检索这些指标的最佳方法是使用 AWS CLI 中的 get-metric-data 或者 get-metric-statistics 命令。
>
> 如果要查看的旧指标有一个具有相似维度的当前指标，则可以查看该当前相似指标，然后选择 Source（源）选项卡，并将指标名称和维度字段更改为所需的指标，此外将时间范围更改为报告指标的时间。

以下步骤可帮助您浏览指标命名空间以查找和查看指标。您还可以使用目标搜索词搜索指标。有关更多信息，请参阅 搜索可用指标。

如果您浏览在 CloudWatch 跨账户可观测性中设置为监控账户的账户，则可以从与该监控账户关联的源账户查看指标。当显示来自源账户的指标时，还会显示其所属账户的 ID 或标签。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

使用控制台按命名空间和维度查看可用指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 选择指标命名空间（例如 EC2 或 Lambda）。
4. 选择指标维度（例如 Per-Instance Metrics（每个实例的指标）或 By Function Name（按函数名称））。
5. Browse（浏览）选项卡显示命名空间中该维度的所有指标。每个指标名称旁边都有一个信息按钮，您可以选择查看包含指标定义的弹出窗口。

如果这是 CloudWatch 跨账户可观测性中的监控账户，您还可以看到与该监控账户关联的源账户中的指标。表中的 Account label（账户标签）和 Account id（账户 ID）列显示每个指标来自哪个账户。

您可执行以下操作：

a. 要对表进行排序，请使用列标题。

b. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。

c. 要按账户筛选，请选择账户标签或账户 ID，然后选择 Add to search（添加到搜索）。

d. 要按资源进行筛选，请选择资源 ID，然后选择 Add to search。

e. 要按指标进行筛选，请选择指标名称，然后选择 Add to search。

6. （可选）要将此图表添加到 CloudWatch 控制面板，请选择 Actions（操作），然后选择 Add to dashboard（添加到控制面板）。

使用 AWS CLI 按账户命名空间、维度或指标查看可用指标

使用 list-metrics 命令列出 CloudWatch 指标。有关发布指标的所有服务的命名空间、指标和维度的列表，请参阅 发布 CloudWatch 指标的 AWS 服务。

以下示例命令将列出 Amazon EC2 的所有指标。

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

下面是示例输出。

```
{
  "Metrics" : [
    ...
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkOut"
```

```
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "CPUUtilization"
    },
    {
        "Namespace": "AWS/EC2",
        "Dimensions": [
            {
                "Name": "InstanceId",
                "Value": "i-1234567890abcdef0"
            }
        ],
        "MetricName": "NetworkIn"
    },
    ...
  ]
}
```

列出指定资源的所有可用指标

以下示例指定 AWS/EC2 命名空间和 InstanceId 维度以仅查看指定实例的结果。

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions
 Name=InstanceId,Value=i-1234567890abcdef0
```

列出所有资源的指标

以下示例指定 AWS/EC2 命名空间和指标名称以仅查看指定指标的结果。

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```

在 CloudWatch 跨账户可观测性中从关联的源账户检索指标

以下示例在监控账户中运行，以便从监控账户和所有关联的源账户检索指标。如果您不添加 --include-linked-accounts，则该命令仅返回监控账户的指标。

```
aws cloudwatch list-metrics --include-linked-accounts
```

在 CloudWatch 跨账户可观测性中从源账户检索指标

以下示例在监控账户中运行，以便从 ID 为 111122223333 的源账户检索指标。

```
aws cloudwatch list-metrics --include-linked-accounts --owning-account "111122223333"
```

搜索可用指标

您可以使用目标搜索词，对您账户中的所有指标进行搜索。之后会返回在命名空间、指标名称或维度内具有匹配结果的指标。

如果这是 CloudWatch 跨账户可观测性中的监控账户，您还可以从与该监控账户关联的源账户中搜索指标。

> **Note**
>
> 控制台中不会显示在过去两周内没有任何新数据点的指标。当您在控制台的 All metrics（全部指标）选项卡的搜索框中键入指标名称或维度名称时，它们也不会显示，并且 list-metrics 命令的结果中不会返回它们。检索这些指标的最佳方法是使用 AWS CLI 中的 get-metric-data 或者 get-metric-statistics 命令。

在 CloudWatch 中搜索可用指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 在 All metrics（所有指标）选项卡上的搜索字段中，输入搜索词（例如，指标名称、命名空间、账户 ID、账户标签、维度名称/值或资源名称）。这将为您显示带有匹配此搜索词的指标的所有命名空间。

   例如，如果搜索 **`volume`**，则将显示其名称中包含该搜索词的指标的命名空间。

   有关搜索的更多信息，请参阅在图表中使用搜索表达式。
4. 要绘制所有搜索结果的图表，请选择 Graph search（图表搜索）

   或者

选择命名空间以查看该命名空间中的指标。然后，您可执行以下操作：

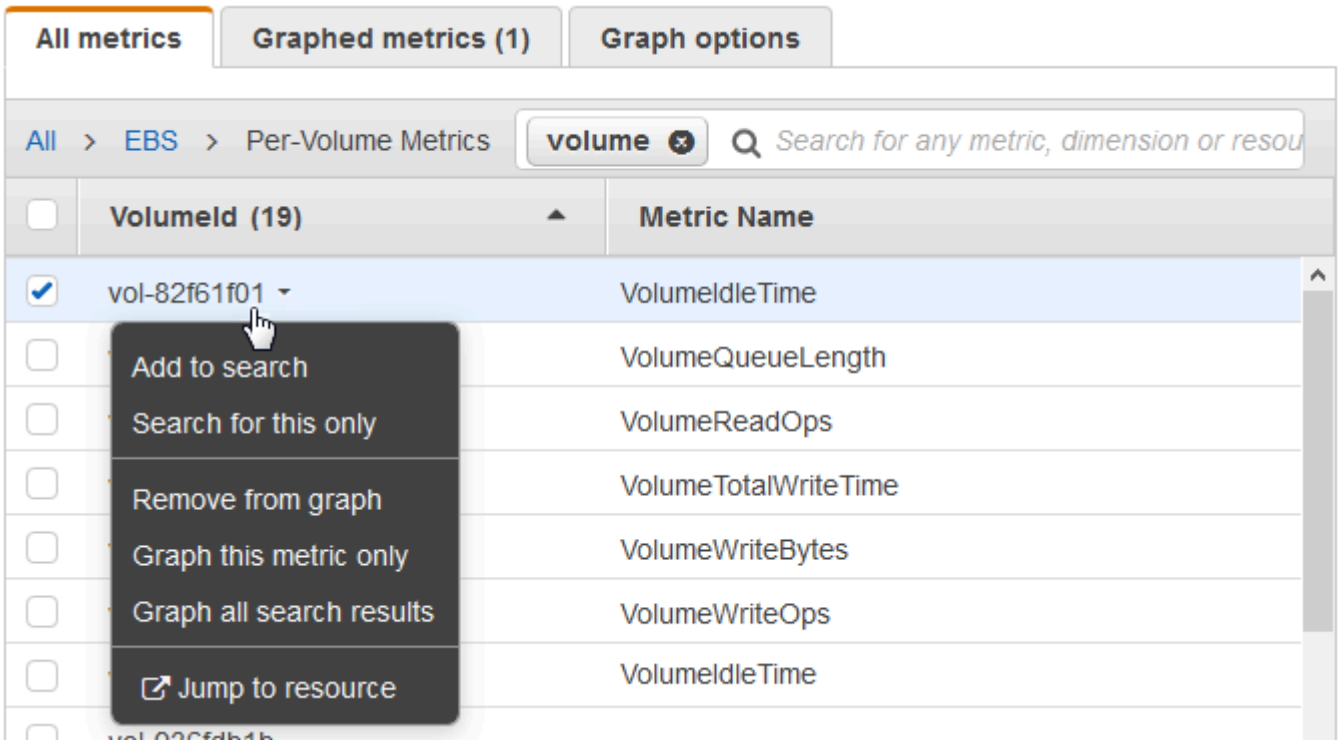
a. 要为一个或多个指标绘制图表，请选中每个指标旁边的复选框。要选择所有指标，请选中表的标题行中的复选框。

b. 要细化您的搜索，请将鼠标指针悬停在某个指标名称上，然后选择 Add to search（添加到搜索）或 Search for this only（仅为此搜索）。

c. 要在控制台中查看某个资源，请选择资源 ID，然后选择 Jump to resource（跳转到资源）。

d. 要查看指标的帮助，请选择指标名称，然后选择 What is this?。

所选指标会显示在图表上。



5. （可选）选择搜索栏中的一个按钮以编辑搜索词的该部分。

绘制指标的图表

使用 CloudWatch 控制台绘制其他 AWS 服务生成的指标数据的图表。这样可以更高效地查看服务上的指标活动。以下程序介绍了如何在 CloudWatch 中绘制指标图表。

内容

- 绘制指标图表

- 将两个图表合并为一个
- 使用动态标签
- 修改图表的时间范围或时区格式
- 放大折线图或堆叠面积图
- 修改图表的 Y 轴
- 从图表上的指标创建告警

绘制指标图表

您可以使用 CloudWatch 控制台选择指标并创建指标数据图表。

CloudWatch 支持指标的以下统计数据：`Average`、`Minimum`、`Maximum`、`Sum` 和 `SampleCount`。有关更多信息，请参阅 统计数据。

您可以在不同的详细程度下查看您的数据。例如，您可以选择一分钟视图，这在您排查问题时会非常有用。或者，选择不太详细的一小时视图。这在您查看更大的时间范围（例如 3 天）来了解一段时间内的趋势时会很有用。有关更多信息，请参阅 时间段。

如果您使用的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则可以用图表表示与该监控账户关联的源账户中的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

创建图表

绘制指标图表

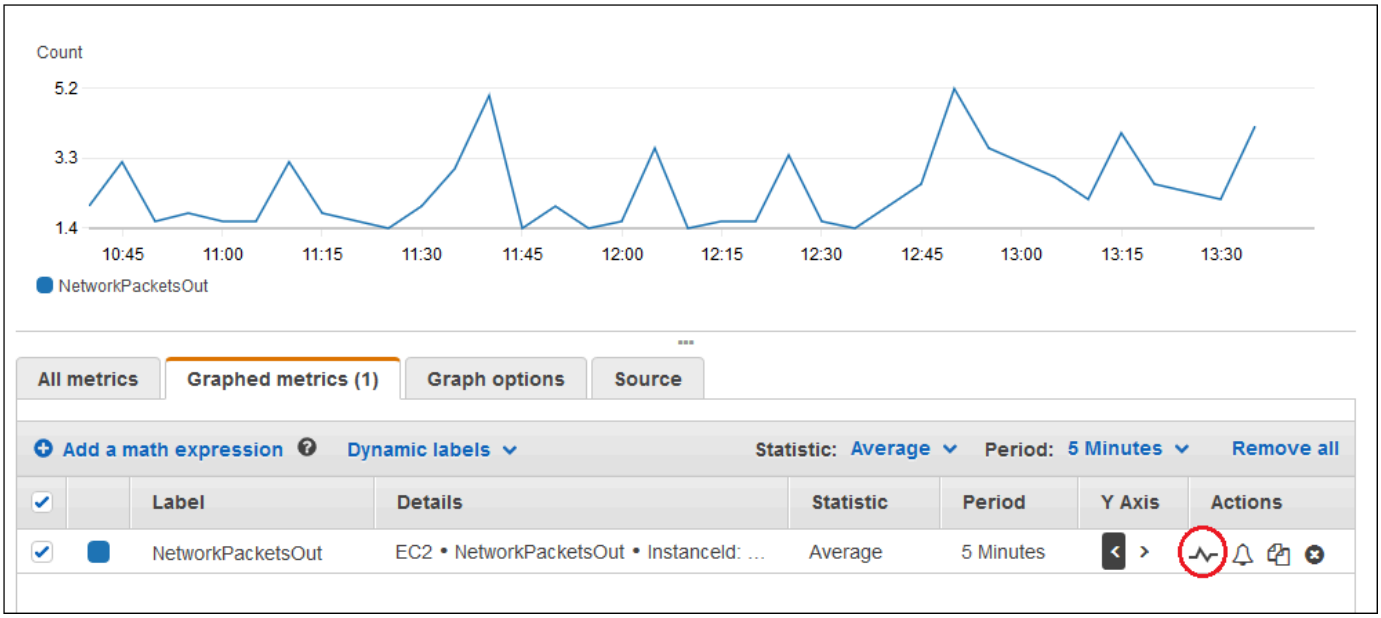
1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 在浏览选项卡上的搜索字段中，输入搜索词（例如，指标名称、账户 ID 或资源名称）。

   例如，如果您搜索 `CPUUtilization` 指标，则将显示具有该指标的命名空间和维度。
4. 选择某个搜索结果以查看指标。
5. 要为一个或多个指标绘制图表，请选中每个指标旁边的复选框。要选择所有指标，请选中表的标题行中的复选框。
6. （可选）要更改图表类型，请选择选项选项卡。然后，您可以从折线图、堆叠面积图、数字显示、量规图、条形图或饼形图中选择。
7. 选择绘成图表的指标选项卡。

8. （可选）若要更改图表中使用的统计数据，请选择指标名称旁边 Statistic（统计数据）列中的新统计数据。

   有关 CloudWatch 统计数据的更多信息，请参阅 CloudWatch 统计数据定义。有关 pxx 百分位数统计数据的更多信息，请参阅 百分位数。

9. （可选）要添加显示指标的预期值的异常检测范围，请选择指标旁边的操作下的异常检测图标。



CloudWatch 使用指标最多两周的最新历史数据来计算预期值的模型。然后将此预期值范围以条带形式显示在图表上。CloudWatch 会在指标下新增一行，以显示异常检测范围数学表达式，并标记为 ANOMALY_DETECTION_BAND。最近的历史数据存在时，您可以立即看到一个预览异常检测范围，该范围是模型生成的异常检测范围的近似值。实际异常检测范围可能需要长达 15 分钟才能显示出来。

预设情况下，CloudWatch 使用范围阈值的默认值 2 创建预期值范围的上限和下限。要更改此数字，请更改范围的 Details (详细信息) 下的公式末尾的值。

- （可选）选择 Edit model (编辑模型) 来更改计算异常检测模型的方式。您可以排除过去和未来的时间段，不在模型计算训练中使用。从训练数据中排除系统中断、部署和假日等异常事件至关重要。您还可以指定要用于夏令时更改模型的时区。

有关更多信息，请参阅 使用 CloudWatch 异常检测。

要在图表中隐藏模型，请在 `ANOMALY_DETECTION_BAND` 函数所在行中取消选中选中标记或者选择 X 图标。要完全删除模型，请选择 Edit model (编辑模型)，然后选择 Delete model (删除模型)。

10. （可选）在选择要绘制图表的指标时，为每个指标指定一个在图表图例上显示的动态标签。动态标签显示有关指标的统计数据，并在刷新控制面板或图表时自动进行更新。要添加动态标签，请选择已绘制图表指标，然后选择添加动态标签。

    默认情况下，添加到标签的动态值显示在标签开头。然后，您可以选择指标的标签值以编辑标签。有关更多信息，请参阅 使用动态标签。

11. 要查看有关要为其绘制图表的指标的更多信息，请将鼠标指针暂停在图例上。

12. 横向注释可帮助图表用户更高效地查看指标的峰值何时达到特定级别以及指标是否在预定义的范围内。要添加横向注释，请选择选项选项卡，然后选择添加横向注释：

    a. 对于 Label (标签)，输入注释的标签。

    b. 对于 Value (值)，输入显示横向注释的指标值。

    c. 对于 Fill，指定是否将填充阴影用于此注释。例如，为要填充的相应区域选择 `Above` 或 `Below`。如果您指定 `Between`，则另一个 `Value` 字段将出现，并且将填充两个值之间的图表区域。

    d. 对于 Axis，指定 `Value` 中的数字是否在图表包含多个指标的情况下表示与左侧 Y 轴或右侧 Y 轴关联的指标。

    您可以通过在注释的左列中选择颜色方块来更改注释的填充色。

    重复这些步骤可向同一个图表添加多个横向注释。

    要隐藏注释，请清除该注释的左列中的复选框。

    要删除注释，请在 Actions 列中选择 x。

13. 要获取图表的 URL，请依次选择 Actions 和 Share。复制要保存或共享的 URL。

14. 要将图表添加到控制面板，请依次选择 Actions 和 Add to dashboard。

创建源自另一个数据来源的指标图表

您可以创建一个图表来显示源自非 CloudWatch 数据来源的资源。有关创建与这些其他数据来源的连接的更多信息，请参阅查询源自其他数据来源的指标。

绘制源自另一个数据来源的指标图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 选择多来源查询选项卡。
4. 对于数据来源，选择要使用的数据来源。

   如果您尚未创建与所需数据来源的连接，请选择创建和管理数据来源，然后选择创建和管理数据来源。有关此数据来源创建过程其余部分的信息，请参阅 通过向导连接到预构建数据来源。
5. 向导或查询编辑器将提示您输入查询所需的信息。每个数据来源的工作流程都不同，并且这些工作流程都是针对数据来源量身定制的。例如，对于 Amazon Managed Service for Prometheus 和 Prometheus 数据来源，系统会显示带有查询助手的 PromQL 查询编辑器框。
6. 完成查询构造后，选择图表查询。

   图表会填充查询中的指标。
7. （可选）横向注释可帮助图表用户更高效地查看指标的峰值何时达到特定级别，以及指标是否在预定义的范围内。要添加横向注释，请选择选项选项卡，然后选择添加横向注释：

   a. 对于 Label (标签)，输入注释的标签。

   b. 对于 Value (值)，输入显示横向注释的指标值。

   c. 对于 Fill，指定是否将填充阴影用于此注释。例如，为要填充的相应区域选择 `Above` 或 `Below`。如果您指定 `Between`，则另一个 `Value` 字段将出现，并且将填充两个值之间的图表区域。

   d. 对于 Axis，指定 `Value` 中的数字是否在图表包含多个指标的情况下表示与左侧 Y 轴或右侧 Y 轴关联的指标。

      您可以通过在注释的左列中选择颜色方块来更改注释的填充色。

   重复这些步骤可向同一个图表添加多个横向注释。

   要隐藏注释，请清除该注释的左列中的复选框。

   要删除注释，请在 Actions 列中选择 x。
8. （可选）要将此图表添加到控制面板，请选择操作，然后选择添加到控制面板。

更新图表

更新图表

1. 要更改图表的名称，请选择铅笔图标。
2. 要更改时间范围，请选择某个预定义的值或选择 custom。有关更多信息，请参阅 修改图表的时间范围或时区格式。
3. 要更改统计数据，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择某个统计数据或预定义百分位数，或指定自定义百分位数（例如 **p95.45**）。
4. 要更改时间段，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择其他值。
5. 要添加横向注释，请选择 Graph options (图表选项)，然后选择 Add horizontal annotation (添加横向注释)：

   a. 对于 Label (标签)，输入注释的标签。
   b. 对于 Value (值)，输入显示横向注释的指标值。
   c. 对于 Fill，指定是否将填充阴影用于此注释。例如，为要填充的相应区域选择 `Above` 或 `Below`。如果您指定 `Between`，则另一个 `Value` 字段将出现，并且将填充两个值之间的图表区域。
   d. 对于 Axis，指定 `Value` 中的数字是否在图表包含多个指标的情况下表示与左侧 Y 轴或右侧 Y 轴关联的指标。

   您可以通过在注释的左列中选择颜色方块来更改注释的填充色。

   重复这些步骤可向同一个图表添加多个横向注释。

   要隐藏注释，请清除该注释的左列中的复选框。

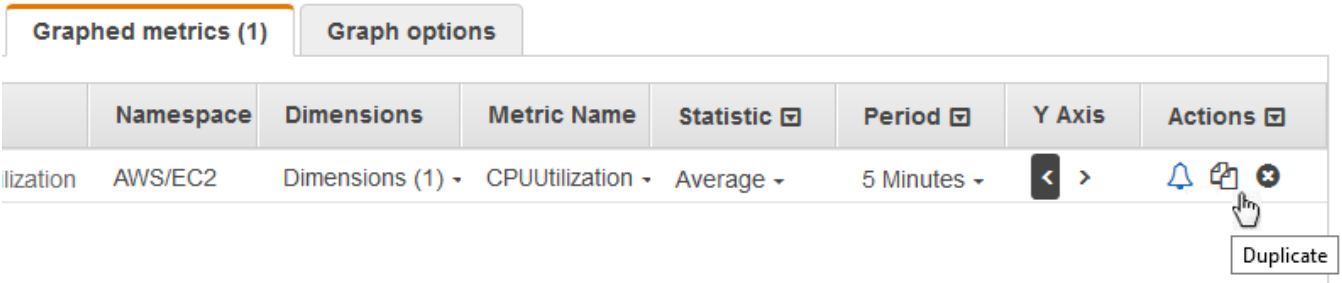
   要删除注释，请在 Actions 列中选择 x。
6. 要更改刷新间隔，请选择 Refresh options (刷新选项)，然后选择 Auto refresh (自动刷新) 或者选择 1 Minute (1 分钟)、2 Minutes (2 分钟)、5 Minutes (5 分钟) 或 15 Minutes (15 分钟)。

复制指标

复制指标

1. 选择 Graphed metrics（已绘制图表指标）选项卡。

2. 对于 Actions，选择 Duplicate 图标。



3. 根据需要更新重复指标。

将两个图表合并为一个

您可以将两个不同的图表合并为一个，然后生成的图表将显示这两个指标。如果您已经在不同的图表中显示了不同的指标并希望将它们合并，或者您想轻松地使用来自不同地区的指标创建单个图表，这会很有用。

要将一个图表合并到其他图表中，您可以使用要合并的图表的 URL 或 JSON 源。

要将两个图表合并为一个

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 打开要合并到其他图表中的图表。为此，您可以选择指标、所有指标，然后选择要生成图表的指标。或者，您可以打开控制面板，然后通过选择图表，并从图表右上角的菜单中选择在指标中打开来打开控制面板上的其中一个图表。
3. 在打开图表之后，请执行以下操作之一：
   - 从浏览器栏复制 URL。
   - 选择资源选项卡，然后选择复制。
4. 打开要合并到上一个图表中的图表。
5. 当您在指标视图中打开第二个图表时，依次选择操作、合并图表。
6. 输入您之前复制的 URL 或 JSON，然后选择合并。
7. 此时将显示合并的图表。左侧的 Y 轴用于原始图表，右侧的 y 轴用于您合并到其中的图表。

> **Note**
>
> 如果您合并的图表使用 METRICS() 函数，则合并后的图表中的指标不包括在合并的图表的 METRICS() 计算中。

8. 要将合并的图表保存到控制面板，请依次选择操作和添加到控制面板。

使用动态标签

您可以在图表中使用动态标签。动态标签在选定指标的标签中添加动态更新的值。您可以向标签添加多种值，如下表所示。

在标签中显示的动态值来自于当前在图表上显示的时间范围。在刷新控制面板或图表时，将自动更新标签的动态部分。

如果将动态标签与搜索表达式一起使用，则动态标签适用于搜索返回的每个指标。

您可以使用 CloudWatch 控制台向标签添加动态值、编辑标签、更改动态值在标签列中的位置以及自定义其他内容。

动态标签

在动态标签中，您可以使用与指标属性相关的以下值：

动态标签实时值	描述
${AVG}	当前在图表中显示的时间范围内的值的平均值。
${DATAPOINT_COUNT}	图表中当前所示时间范围内的数据点的数量。
${FIRST}	图表中当前所示时间范围内的最早指标值。
${FIRST_LAST_RANGE}	图表中当前所示最早数据点和最新数据点的指标值之间的差值。
${FIRST_LAST_TIME_RANGE}	图表中当前所示最早数据点和最新数据点之间的绝对时间范围。
${FIRST_TIME}	图表中当前所示时间范围内的最早数据点的时间戳。
${FIRST_TIME_RELATIVE}	此刻与图表中当前所示时间范围内的最早数据点的时间戳之间的绝对时间差。
${LABEL}	指标的默认标签的表示形式。
${LAST}	图表中当前所示时间范围内的指标值的最新值。

动态标签实时值	描述
${LAST_TIME}	图表中当前所示时间范围内的最新数据点的时间戳。
${LAST_TIME_RELATIVE}	此刻与图表中当前所示时间范围内的最新数据点的时间戳之间的绝对时间差。
${MAX}	当前在图表中显示的时间范围内的值的最大值。
${MAX_TIME}	图表中当前所示数据点中具有最高指标值的数据点的时间戳。
${MAX_TIME_RELATIVE}	此刻与图表中当前所示数据点中具有最高值的数据点的时间戳之间的绝对时间差。
${MIN}	当前在图表中显示的时间范围内的值的最小值。
${MIN_MAX_RANGE}	图表中当前所示数据点中具有最高指标值和最低指标值的数据点之间的指标值之差。
${MIN_MAX_TIME_RANGE}	图表中当前所示数据点中具有最高指标值和最低指标值的数据点之间的绝对时间范围。
${MIN_TIME}	图表中当前所示数据点中具有最低指标值的数据点的时间戳。
${MIN_TIME_RELATIVE}	此刻与图表中当前所示数据点中具有最低值的数据点的时间戳之间的绝对时间差。
${PROP('AccountId')}	指标的 AWS 账户 ID。
${PROP('AccountLabel')}	在 CloudWatch 跨账户可观测性中,为拥有该指标的源账户指定的标签。
${PROP('Dim.*dimension_name*')}	指定维度的值。将 *dimension_name* 替换为您的维度名称（区分大小写。
${PROP('MetricName')}	指标的名称。
${PROP('Namespace')}	指标的命名空间。
${PROP('Period')}	指标的时间段（以秒为单位）。

动态标签实时值	描述
${PROP('Region')}	发布指标的 AWS 区域。
${PROP('Stat')}	正在用于绘制图表的指标统计数据。
${SUM}	当前在图表中显示的时间范围内的值的总和。

例如，假设您使用 **`SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum')`** 搜索表达式，它查找每个 Lambda 函数的 `Errors`。如果将标签设置为 `[max: ${MAX} Errors for Function Name ${LABEL}]`，则每个指标的标签为 [max: ***`number`*** Errors for Function Name ***`Name`***]。

您最多可以在标签中添加 6 个动态值。您只能在每个标签中使用一次 `${LABEL}` 占位符。

修改图表的时间范围或时区格式

本节介绍了如何在 CloudWatch 指标图表上修改日期、时间和时区格式。它还介绍了如何放大图表以应用特定的时间范围。有关创建图表的信息，请参阅 绘制指标图表。

> **Note**
>
> 如果控制面板的时间范围短于其上用于图表的时间段，则会发生以下情况：
>
> - 图表会被修改以显示该小组件的一个完整时间段对应的数据量，即使该时间段长于控制面板时间范围，也是如此。这样可以确保图表上至少有一个数据点。
> - 该数据点时间段的开始时间会被向后调整，以确保至少可以显示一个数据点。

设置相对时间范围

New interface

指定图表的相对时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12

小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 Custom（自定义）来设置自己的时间范围。

3. 选择 Custom（自定义），然后选择框的左上角的 Relative（相对）选项卡。您可以以 Minutes（分钟）、Hours（小时）、Days（天）、Weeks（周）、Months（月）为单位指定时间范围。
4. 在指定时间范围后，选择 Apply（应用）。

Original interface

指定图表的相对时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 custom（自定义）来设置自己的时间范围。
3. 选择 custom（自定义），然后选择框的左上角的 Relative（相对）。您可以以 Minutes（分钟）、Hours（小时）、Days（天）、Weeks（周）或 Months（月）为单位指定时间范围。

设置绝对时间范围

New interface

指定图表的绝对时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 Custom（自定义）来设置自己的时间范围。
3. 选择 Custom（自定义），然后选择框的左上角的 Absolute（绝对）选项卡。使用日历选取器或文本字段框指定时间范围。
4. 在指定时间范围后，选择 Apply（应用）。

Original interface

指定图表的绝对时间范围

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 custom（自定义）来设置自己的时间范围。
3. 选择 custom（自定义），然后选择框的左上角的 Absolute（绝对）。使用日历选取器或文本字段框指定时间范围。
4. 在指定时间范围后，选择 Apply（应用）。

设置时区格式

New interface

指定图表的时区

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 Custom（自定义）来设置自己的时间范围。
3. 选择 Custom（自定义），然后选择框右上角的下拉菜单。您可以将时区更改为 UTC（协调世界时）或 Local time zone（本地时区）。
4. 在更改之后，选择 Apply（应用）。

Original interface

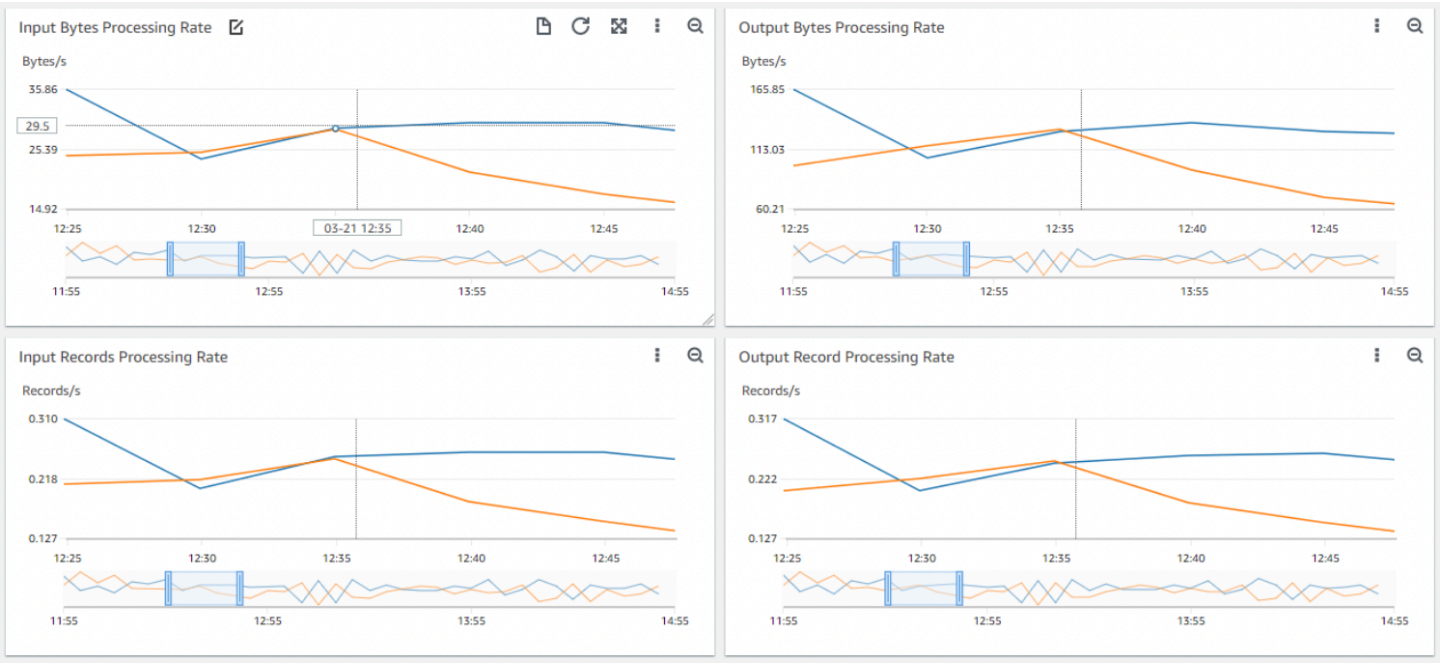
指定图表的时区

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。在屏幕右上角，您可以选择从 1 小时到 1 周的某个预定义的时间范围（1h（1 小时）、3h（3 小时）、12h（12 小时）、1d（1 天）、3d（3 天）或者 1w（1 周））。或者，您也可以选择 custom（自定义）来设置自己的时间范围。

3. 选择 custom（自定义），然后选择框右上角的下拉菜单。您可以将时区更改为 UTC（协调世界时）或 Local timezone（本地时区）。

放大折线图或堆叠面积图

在 CloudWatch 控制台中，您可以使用缩微贴图缩放功能来重点查看折线图和堆叠面积图的各个部分，而无需在放大和缩小视图之间进行切换。例如，您可以使用缩微贴图缩放功能来重点查看折线图中的峰值，以便将该峰值与同一时间线的控制面板中的其他指标进行比较。本节内容中的程序描述如何使用缩放功能。



在上图中，缩放功能突出了与输入字节处理速率相关的折线图的峰值，它同时还在控制面板中显示其他折线图，突出同一时间线的各个部分。

New interface

放大图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。
3. 选择 Browse（浏览），然后选择要生成图表的一个或多个指标。
4. 选择 Options（选项），然后选择 Widget type（小组件类型）下方的 Line（折线图）。
5. 选择并拖动要突出的图表区域，然后释放拖动对象。

6. 要重置缩放，选择 Reset zoom（重置缩放）图标，该图标看起来像放大镜里面包含减号 (-) 符号。

Original interface

放大图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。
3. 选择 All metrics（所有指标），然后选择要生成图表的指标。
4. 选择 Graph options（图表选项）。在 Widget type（小组件类型）下方，选择 Line（折线图）。
5. 选择并拖动要突出的图表区域，然后释放拖动对象。
6. 要重置缩放，选择 Reset zoom（重置缩放）图标，该图标看起来像放大镜里面包含减号 (-) 符号。

> **Tip**
>
> 如果您已创建包含折线图或堆叠面积图的控制面板，可以转至控制面板并开始使用缩放功能。

修改图表的 Y 轴

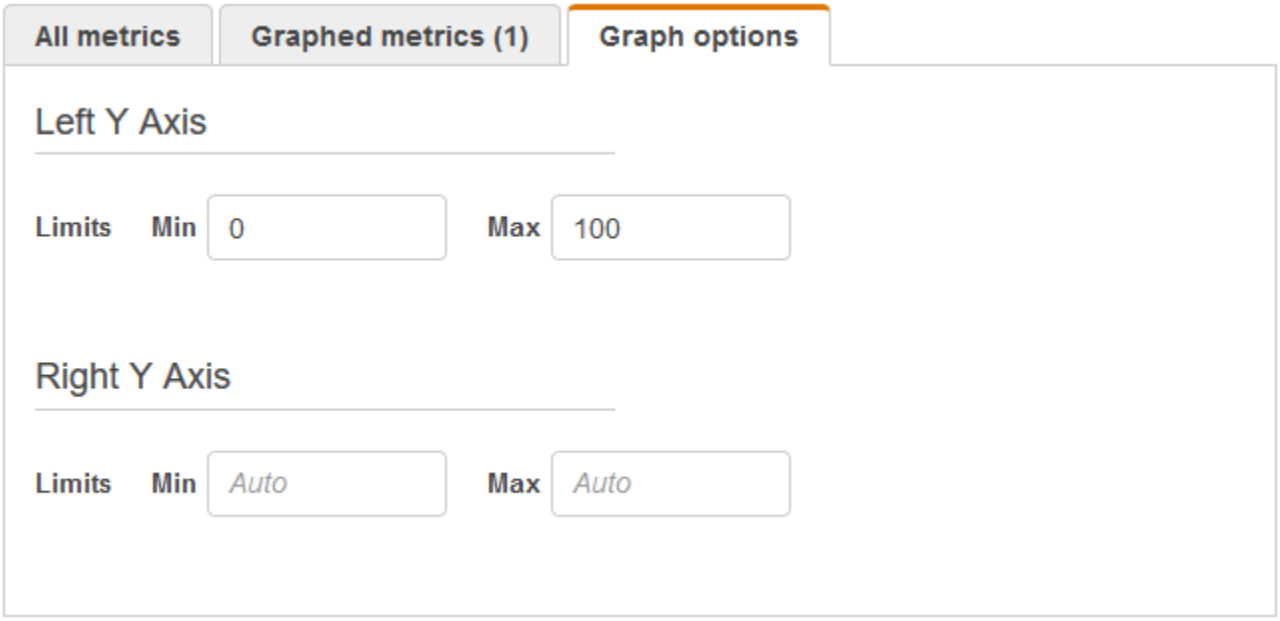
您可以在图表上设置 Y 轴的自定义范围，帮助您更好地查看数据。例如，您可以将 `CPUUtilization` 图表上的范围更改为 100%，这样，您就能轻松看到 CPU 使用率是低（绘制的线条靠近图表底部）还是高（绘制的线条靠近图表顶部）。

您可以在图表的两个不同的 Y 轴之间切换。这在图表包含单位不同或者值范围差异很大的指标时非常有用。

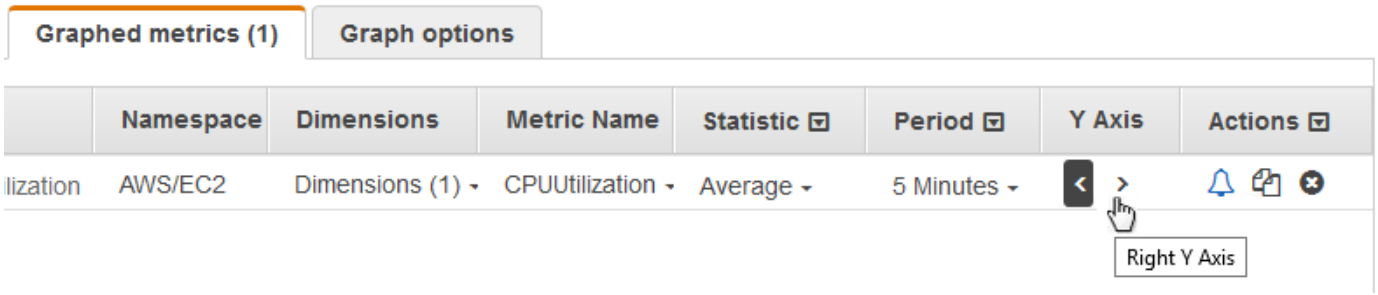
修改图表上的 Y 轴

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择一个指标命名空间（例如 EC2），然后选择一个指标维度（例如 Per-Instance Metrics (每个实例的指标)）。

4. All metrics 选项卡显示此命名空间中该维度的所有指标。要为指标绘制图表，请选中该指标旁的复选框。
5. 在 Graph options（绘制图表选项）选项卡上，指定 Left Y Axis（左侧 Y 轴）的 Min（最小值）和 Max（最大值）。Min (最小值) 的值不能大于 Max (最大值) 的值。



6. 要创建第二个 Y 轴，请指定 Right Y Axis（右侧 Y 轴）的 Min（最小值）和 Max（最大值）。
7. 要在两个 Y 轴之间切换，请选择 Graphed metrics（已绘制图表指标）选项卡。对于 Y Axis，请选择 Left Y Axis 或 Right Y Axis。



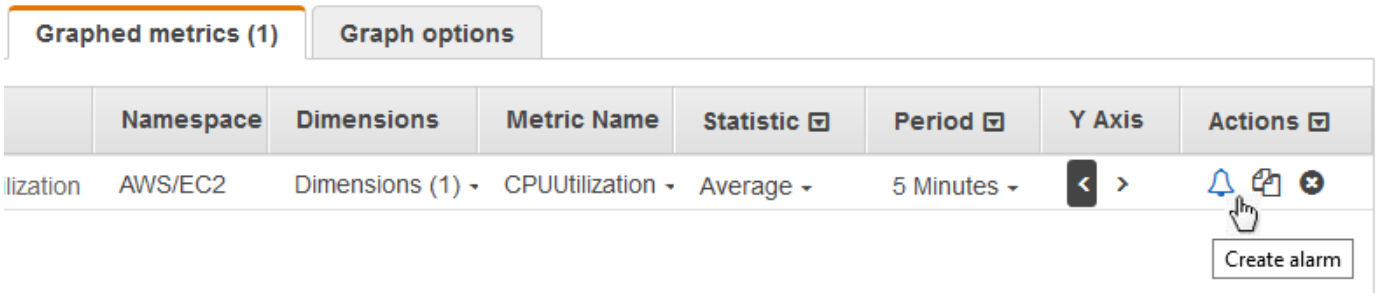
从图表上的指标创建告警

您可以绘制指标的图表，然后从图表上的指标创建警报，这样做的优点是可为您填充很多警报字段。

从图表上的指标创建警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。

3. 选择一个指标命名空间（例如 EC2），然后选择一个指标维度（例如 Per-Instance Metrics (每个实例的指标)）。
4. All metrics 选项卡显示此命名空间中该维度的所有指标。要为指标绘制图表，请选中该指标旁的复选框。
5. 要为指标创建警报，请选择 Graphed metrics（已绘制图表指标）选项卡。对于 Actions，选择警报图标。



6. 在条件下，选择静态或异常检测，以指定对警报使用静态阈值还是异常检测模型。

   根据您的选择，输入警报条件的其余数据。
7. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

   要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。
8. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
9. 选择下一步。
10. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

    要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

    要让警报不发送通知，请选择删除。
11. 要让警报执行 Auto Scaling 或 EC2 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。
12. 在完成后，选择下一步。
13. 输入警报的名称和说明。名称只能包含 ASCII 字符。然后选择下一步。
14. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

使用 CloudWatch 异常检测

为指标启用异常检测时，CloudWatch 将应用统计算法和机器学习算法。这些算法只需最少的用户干预，即可持续分析系统和应用程序的指标，确定正常基线和表面异常。

这些算法会生成异常检测模型。该模型生成表示正常指标行为的预期值范围。

您可以使用 AWS Management Console、AWS CLI、AWS CloudFormation 或 AWS SDK 启用异常检测。您可以对由 AWS 出售的指标以及自定义指标启用异常检测。在设置为 CloudWatch 跨账户可观测性的账户中，除了监控账户中的指标外，您还可以在源账户的指标中创建异常检测器。

您可以通过两种方式使用预期值模型：

- 根据指标的预期值创建异常检测警报。这些类型的警报没有用于确定警报状态的静态阈值。相反，它们将根据异常检测模型将指标的值与预期值进行比较。

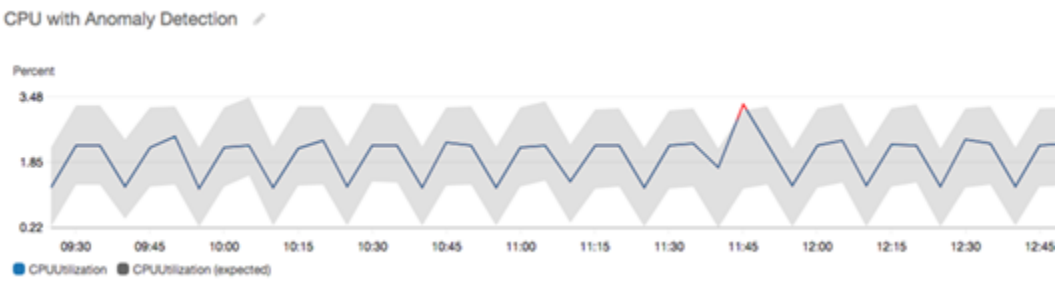
  您可以选择当指标值高于预期值范围和/或低于预期值范围时是否触发警报。

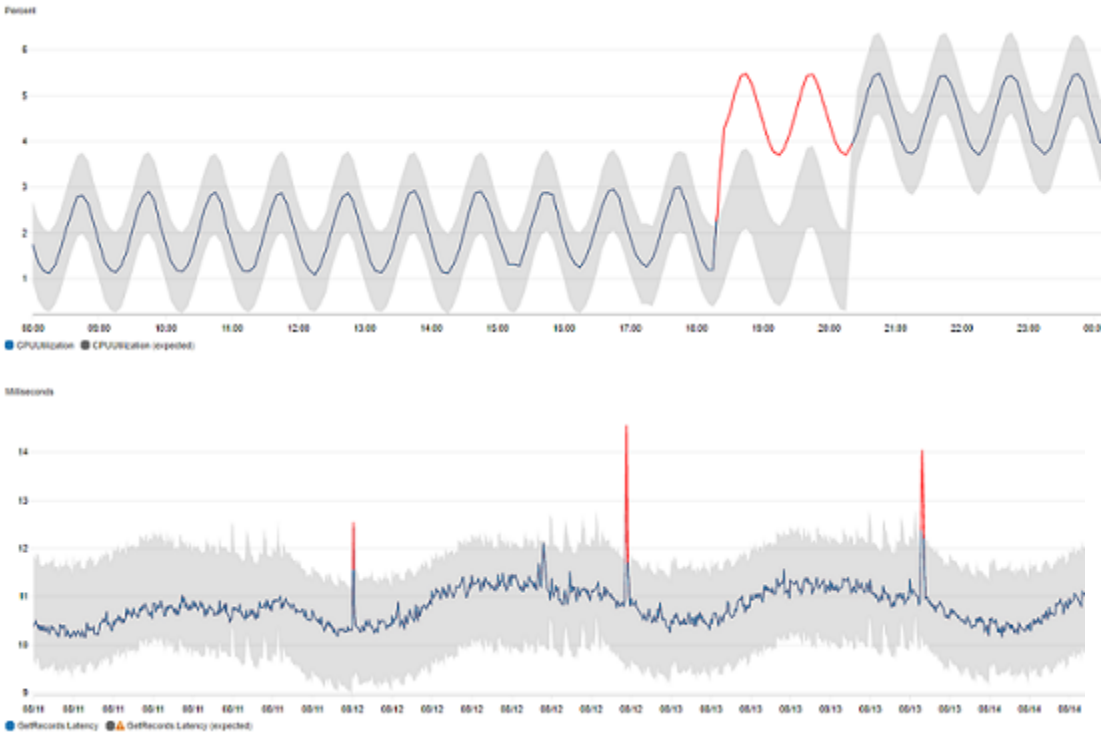
  有关更多信息，请参阅 根据异常检测创建 CloudWatch 告警。
- 查看指标数据的图表时，将预期值叠加到图表上作为范围。这样可以清晰直观地看出图中的哪些值不在正常范围内。有关更多信息，请参阅 创建图表。

  您还可以通过将 `GetMetricData` API 请求与 `ANOMALY_DETECTION_BAND` 指标数学函数结合使用来检索模型范围的上限值和下限值。有关更多信息，请参阅 GetMetricData。

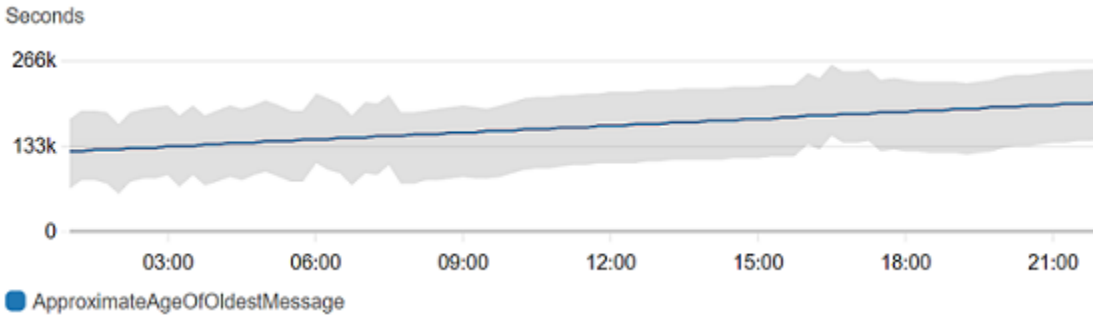
在具有异常检测的图表中，预期的值范围显示为灰色范围。如果指标的实际值超出此范围，则在此期间将显示为红色。

异常检测算法将指标的季节性变化和趋势变化考虑在内。季节性变化可以是每小时、每天或每周，如以下示例所示。

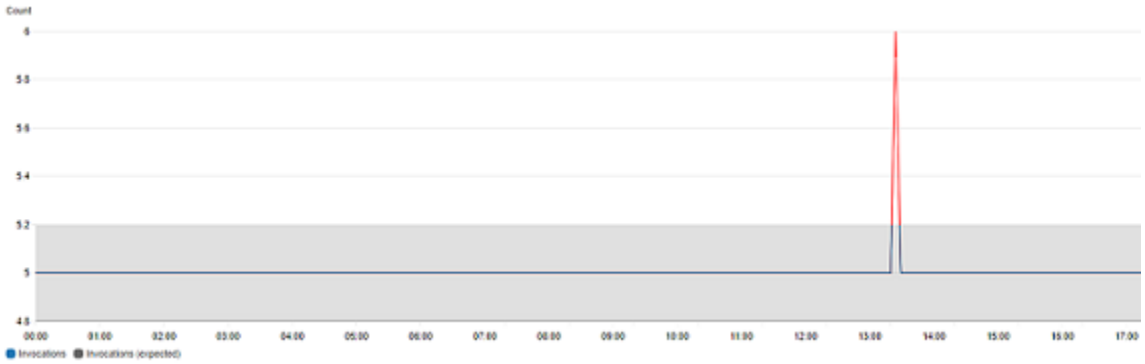




较长期的趋势可能是向下或向上。



异常检测也适用于具有平面模式的指标。



CloudWatch 异常检测的工作原理

在为指标启用异常检测后，CloudWatch 会将机器学习算法应用于指标的过去数据，以创建指标的预期值模型。该模型评估指标的趋势以及每小时、每日和每周模式。算法训练最多两周的指标数据，但即使指标没有完整的两周数据，您也可以为指标启用异常检测。

您需要为异常检测阈值指定一个值，CloudWatch 使用该值和该模型确定指标值的“正常”范围。异常检测阈值的值越高，生成的“正常”值范围越大。

机器学习模型特定于指标和统计数据。例如，如果您使用 AVG 统计数据为指标启用异常检测，则模型特定于 AVG 统计数据。

当 CloudWatch 为许多 AWS 服务中的常见指标创建模型时，它可确保检测范围不会超出逻辑值。例如，EC2 实例 MemoryUtilization 的波段将保持在 0 到 100 之间，而跟踪 CloudFront Requests 的波段（不能为负）永远不会延伸到零以下。

创建模型后，CloudWatch 异常检测会持续评估模型并对其进行调整，以确保模型尽可能准确。这包括重新训练模型，以便在指标值随时间推移而变化或有突然变化时进行调整，还包括用于改进季节性、峰值或稀疏指标模型的预测器。

在为指标启用异常检测后，您可以选择排除指标的指定时间段，使其不用于训练模型。这样，您就可以排除部署或其他不寻常的事件，使其不用于训练模型，从而确保创建最精确的模型。

对警报使用异常检测模型会使您的 AWS 账户产生费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

指标数学异常检测

指标数学异常检测是一项您可以用于针对指标数学表达式的输出创建异常检测告警的功能。您可以使用这些表达式来创建能可视化异常检测范围的图表。该功能支持基本的算术函数、比较和逻辑运算符以及大多数其他函数。有关不支持的函数的信息，请参阅 Amazon CloudWatch 用户指南中的使用指标数学。

您可以根据指标数学表达式创建异常检测模型，创建方式类似于创建异常检测模型。在 CloudWatch 控制台中，您可以将异常检测应用于指标数学表达式，并选择异常检测作为这些表达式的阈值类型。

> **Note**
>
> 仅可在最新版本的指标用户界面中启用和编辑指标数学异常检测。在新版本指标用户界面中根据指标数学表达式创建异常检测器时，您可以在旧版本中查看这些检测器，但不能对它们进行编辑。

有关如何为异常检测和指标数学创建告警和模型的信息，请参阅以下部分：

- 根据异常检测创建 CloudWatch 告警
- 根据指标数学表达式创建 CloudWatch 告警

您还可以将 CloudWatch API 与 `PutAnomalyDetector`、`DeleteAnomalyDetector` 和 `DescribeAnomalyDetectors` 结合起来使用，根据指标数学表达式创建、删除及发现异常检测模型。有关这些 API 操作的信息，请参阅 Amazon CloudWatch API 参考中的以下部分。

- PutAnomalyDetector
- DeleteAnomalyDetector
- DescribeAnomalyDetectors

有关异常检测告警定价方式的信息，请参阅 Amazon CloudWatch 定价。

使用指标数学

通过指标数学可以查询多个 CloudWatch 指标，可以使用数学表达式基于这些指标创建新的时间序列。您可以在 CloudWatch 控制台上直观显示生成的时间序列，并将其添加到控制面板中。以 AWS Lambda 指标为例，您可以将 `Errors` 指标除以 `Invocations` 指标来获得错误率。然后，将生成的时间序列添加到 CloudWatch 控制面板上的图表中。

您也可以使用 `GetMetricData` API 操作以编程方式执行指标数学。有关更多信息，请参阅 GetMetricData。

向 CloudWatch 图表中添加数学表达式

您可以向 CloudWatch 控制面板上的图表中添加数学表达式。每个图表限制为使用最多 500 个指标和表达式，因此仅当图表具有 499 个或更少的指标时，您才可以添加数学表达式。即使并非所有指标都显示在图表上，这一点也适用。

向图表中添加数学表达式

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 创建或编辑图表。图表中至少需要一个指标。
3. 选择 Graphed metrics（已绘制图表指标）。
4. 选择 Math expression (数学表达式)、Start with empty expression (从空表达式开始)。将为表达式显示一个新行。
5. 在 Details (详细信息) 列下方的新行中，输入数学表达式。指标数学语法和函数部分中的表列出了可以在表达式中使用的函数。

   要使用一个指标或另一个表达式的结果作为此表达式的公式的一部分，请使用 Id 列中显示的值：例如，m1+m2 或 e1-MIN(e1)。

   您可以更改 Id 的值。它可以包括数字、字母和下划线，并且必须以小写字母开头。将 Id 的值更改为更有意义的名称也可以使图表更易于理解；例如，从 m1 和 m2 更改为 errors 和 requests。

   > **Tip**
   >
   > 选择 Math Expression (数学表达式) 旁边的向下箭头可查看受支持的函数列表，您可以在创建表达式时使用这些函数。

6. 对于表达式的 Label (标签) 列，输入一个描述表达式正在计算的内容的名称。

   如果某个表达式的结果是一组时间序列，则这些时间序列将在图表上单独的行中以不同的颜色显示。图表的下方是图表中的每个行的图例。对于生成多个时间序列的单个表达式，这些时间序列的图例文字的格式为 ***Expression-Label Metric-Label***。例如，如果图表包含一个具有标签 Errors 的指标和一个具有标签 Filled With 0: 的表达式 FILL(METRICS(), 0)，则图例中的一行将为 Filled With 0: Errors。要使图例仅显示原始指标标签，请将 *Expression-Label* 设置为空。

   当一个表达式在图表上生成了一组时间序列时，您无法更改用于这些时间序列的颜色。

7. 在添加所需的表达式后，您可以通过隐藏某些原始指标来简化图表。要隐藏某个指标或表达式，请清除 Id 字段左侧的复选框。

指标数学语法和函数

以下各部分解释可用于指标数学的函数。所有函数都必须用大写字母编写（例如 AVG），所有指标和数学表达式的 Id 字段都必须以小写字母开头。

任何数学表达式的最终结果都必须是单个时间序列或一组时间序列。某些函数会生成标量数字。您可以在一个更大的函数中使用这些函数，从而最终生成一个时间序列。例如，采用单个时间序列的 AVG 会生成标量数字，因此它不能是最终的表达式结果。但您可以在函数 m1-AVG(m1) 中使用它，以显示每个单独数据点与时间序列中平均值之差的时间序列。

数据类型缩写

某些函数仅对某些类型的数据有效。在函数表中使用以下列表中的缩写来代表每个函数支持的数据类型：

- S 代表标量数字，例如 2、-5 或 50.25。
- TS 是时间序列（单个 CloudWatch 指标随时间变化的一系列值）：例如，实例 `i-1234567890abcdef0` 在过去三天的 `CPUUtilization` 指标。
- TS [] 是一个时间序列数组，例如多个指标的时间序列。
- String[] 是一个字符串数组。

METRICS() 函数

METRICS() 函数将返回请求中的所有指标。数学表达式不包括在内。

您可以在一个更大的表达式中使用 METRICS()，从而最终生成单个时间序列或一组时间序列。例如，表达式 SUM(METRICS()) 将返回作为所有绘成图表的指标值的总和的时间序列 (TS)。METRICS()/100 将返回一组时间序列，其中的每个时间序列都显示其中一个指标的各个数据点除以 100 的结果。

您可以将 METRICS() 函数与一个字符串一起使用，仅返回在其 Id 字段中包含该字符串的绘成图表的指标。例如，表达式 SUM(METRICS("errors")) 一个是所有在其 Id 字段中具有“errors”的绘成图表的指标值的总和的时间序列。您还可以使用 SUM([METRICS(“4xx”), METRICS(“5xx”)]) 来匹配多个字符串。

基本算术函数

下表列出了受支持的基本算术函数。时间序列中缺少的值被视为 0。如果数据点的值导致函数试图除以零，则会丢弃该数据点。

操作	参数	示例
算术运算符: + - * / ^	S, S	PERIOD(m1)/60

操作	参数	示例
	S, TS	5 * m1
	TS, TS	m1 - m2
	S, TS[]	SUM(100/[m1, m2])
	TS, TS[]	AVG(METRICS())
		METRICS()*100
一元减法 -	S	-5*m1
	TS	-m1
	TS[]	SUM(-[m1, m2])

比较运算符和逻辑运算符

您可以将比较运算符和逻辑运算符与一对时间序列或一对单标量值结合使用。在将比较运算符与一对时间序列结合使用时，运算符将返回一个时间序列，其中每个数据点为 0 (false) 或 1 (true)。如果在一对标量值上使用比较运算符，则将返回一个单标量值（0 或 1）。

如果在两个时间序列之间使用比较运算符，并且仅一个时间序列具有特定时间戳值，则该函数会将另一个时间序列中的缺失值视为 0。

您可以将逻辑运算符与比较运算符结合使用来创建更复杂的函数。

下表列出了受支持的运算符。

运算符类型	支持的运算符
比较运算符	==
	!=
	<=
	>=
	<

运算符类型	支持的运算符
	>
逻辑运算符	AND 或 && OR 或 ||

为了说明如何使用这些运算符，假设我们有两个时间序列：metric1 具有值 `[30, 20, 0, 0]`，metric2 具有值 `[20, -, 20, -]`，其中 `-` 指示未提供该时间戳的值。

Expression	输出
(metric1 < metric2)	0, 0, 1, 0
(metric1 >= 30)	1, 0, 0, 0
(metric1 > 15 AND metric2 > 15)	1, 0, 0, 0

指标数学支持的函数

下表描述了可在数学表达式中使用的函数。用大写字母输入所有函数。

任何数学表达式的最终结果都必须是单个时间序列或一组时间序列。以下部分中的表中的某些函数会生成标量数字。您可以在一个更大的函数中使用这些函数，从而最终生成一个时间序列。例如，采用单个时间序列的 AVG 会生成标量数字，因此它不能是最终的表达式结果。但您可以在函数 m1-AVG(m1) 中使用它来显示每个数据点和该数据点的平均值之间的差异的时间序列。

在下表中，Examples (示例) 列中的每个示例都是一个表达式，用于生成单个时间序列或一组时间序列。这些示例说明如何将返回标量数字的函数用作生成单个时间序列的有效表达式的一部分。

函数	参数	返回类型*	描述	示例	支持跨账户？
ABS	TS TS[]	TS TS[]	返回每个数据点的绝对值。	ABS(m1-m2) MIN(ABS([m1, m2]))	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
				ABS(METRICS())	
ANOMALY_DETECTION_BAND	TS TS、S	TS[]	返回指定指标的异常检测范围。该范围由两个时间序列组成，一个表示指标的“正常”预期值的上限，另一个表示下限。该函数可接受两个参数。第一个参数是要为其创建范围的指标的 ID。第二个参数是要用于范围的标准差的数目。如果您不指定此参数，则使用默认值 2。有关更多信息，请参阅 使用 CloudWatch 异常检测。	ANOMALY_DETECTION_BAND(m1) ANOMALY_DETECTION_BAND(m1,4)	

函数	参数	返回类型*	描述	示例	支持跨账户？
AVG	TS TS[]	S TS	单个时间序列的 AVG 将返回一个标量，表示指标中所有数据点的平均值。一组时间序列的 AVG 将返回单个时间序列。缺少的值被视为 0。 **Note** 如果您希望函数返回一个标量，则建议您不要在 CloudWatch 警报中使用此函数。例如，AVG(m2)。每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。要将此功能与警报配合使用，尤其是包含自动扩缩操作的警报，我们建议您将警报设置为使用 N 个数据点中的	SUM([m1,m2])/AVG(m2) AVG(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
			M 个，其中 M < N。		
CEIL	TS TS[]	TS TS[]	返回每个指标的上限。上限是大于或等于每个值的最小整数。	CEIL(m1) CEIL(METRICS()) SUM(CEIL(METRICS()))	✓
DATAPOINT_COUNT	TS TS[]	S TS	返回报告了值的数据点的计数。这对于计算稀疏指标的平均值非常有用。 **Note** 我们建议您不要在 CloudWatch 告警中使用此函数。每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	SUM(m1)/DATAPOINT_COUNT(m1) DATAPOINT_COUNT(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
DB_PERF_INSIGHTS	字符串, 字符串, 字符串 String, String, String[]	TS（如果给定单个字符串） TS[]（如果给定字符串数组）	返回 Amazon Relational Database Service 和 Amazon DocumentDB（与 MongoDB 兼容）等数据库的性能详情计数器指标。此函数返回的数据量与您直接查询性能详情 API 所获得的数据量相同。您可以在 CloudWatch 中使用这些指标来绘制图表和创建警报。 **Important** 使用此函数时，必须指定数据库的唯一数据库资源 ID。这与数据库标识符不同。要在 Amazon RDS 控制台中查找数据库资源 ID，请选择数据库实例以查看其详细信息。然后，选择配置选项卡。资源 ID 将显示在配置部分中。	DB_PERF_INSIGHTS(‘RDS’, ‘db-ABCDEFGHIJKLMNOPQRSTUVWXY1’, ‘os.cpuUtilization.user.avg’) DB_PERF_INSIGHTS(‘DOCDB, ‘db-ABCDEFGHIJKLMNOPQRSTUVWXY1’, [‘os.cpuUtilization.idle.avg’, ‘os.cpuUtilization.user.max’])	

函数	参数	返回类型*	描述	示例	支持跨账户？
			DB_PERF_INSIGHTS 还以亚分钟为间隔引入 DBLoad 指标。 使用此函数检索的性能详情指标不会存储在 CloudWatch 中。因此，某些 CloudWatch 功能（例如跨账户可观测性、异常检测、指标流、Metrics Explorer 和 Metric Insights）不适用于您通过 DB_PERF_INSIGHTS 检索的性能详情指标。 使用 DB_PERF_INSIGHTS 函数的单个请求可以检索以下数量的数据点。 • 高精度周期（1 秒、10 秒、30 秒）为 1080 个数据点 • 标准精度周期（1 分钟、5 分钟、1 小时、1 天）为 1440 个数据点 DB_PERF_INSIGHTS 函数仅支持以下周期长度： • 1 秒		

函数	参数	返回类型*	描述	示例	支持跨账户？
			• 10 秒 • 30 秒 • 1 minute • 5 分钟 • 1 小时 • 1 天 有关 Amazon RDS 性能详情计数器指标的更多信息，请参阅性能详情计数器指标。 有关 Amazon DocumentDB 性能详情计数器指标的更多信息，请参阅性能详情计数器指标。 Note DB_PERF_INSIGHTS 检索的精度为亚分钟的高分辨率指标仅适用于 DBLoad 指标，或者如果您启用了更高分辨率的增强监控，则适用于操作系统指标。有关 Amazon RDS 增		

函数	参数	返回类型*	描述	示例	支持跨账户？
			强监控的更多信息，请参阅使用增强监控监控操作系统指标。您可以使用 DB_PERF_INSIGHTS 函数创建最长时间范围为三小时的高分辨率警报。您可以使用 CloudWatch 控制台绘制任何时间范围内通过 DB_PERF_INSIGHTS 函数检索到的指标的图表。		
DIFF	TS TS[]	TS TS[]	返回时间序列中各值与该时间序列中的先前值之间的差值。	DIFF(m1)	✓
DIFF_TIME	TS TS[]	TS TS[]	返回时间序列中各值的时间戳与该时间序列中的先前值的时间戳之间的差值（以秒为单位）。	DIFF_TIME(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
FILL	TS, [S | REPEA | LINEAR TS[], [TS | S | REPEA | LINEAR	TS TS[]	填充时间序列的缺失值。有几个选择可用作缺失值的填充内容： • 您可以指定要用作填充值的值。 • 您可以指定要用作填充值的指标。 • 您可以使用 REPEAT 关键字，以使用缺失值之前的指标的最新实际值来填充缺失值。 • 您可以使用 LINEAR 关键字，以使用在缺口的开头和结尾的值之间创建线性插值的值来填充缺失值。 **Note** 当您在告警中使用此函数时，如果您的指标发布稍有延迟，且最近一分钟从未有数据，则可能会出现问题。在这种情况下，FILL 将会以请求的值来替代缺失的数据点。这	FILL(m1,10) FILL(METRICS(), 0) FILL(METRICS(), m1) FILL(m1, MIN(m1)) FILL(m1, REPEAT) FILL(METRICS(), LINEAR)	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
			会导致指标的最新数据点始终为 FILL 值，从而导致告警卡在 OK（正常）或 ALARM（告警）状态。您可以通过使用 M 个告警（共 N 个）来解决此问题。有关更多信息，请参阅 评估告警。		
FIRST LAST	TS[]	TS	返回时间序列数组中的第一个或最后一个时间序列。这在与 SORT 函数结合使用时非常有用。它还可用于从 ANOMALY_DETECTION_BAND 函数中获取高阈值和低阈值。	IF(FIRST(SORT(METRICS(), AVG, DESC))>100, 1, 0) 查看数组中的最高指标，该指标通过 AVG 进行排序。然后，它为每个数据点返回 1 或 0，具体取决于相应数据点的值是否大于 100。 LAST(ANOMALY_DETECTION_BAND(m1)) 将返回异常预测带的上限。	✓
FLOOR	TS TS[]	TS TS[]	返回每个指标的下限。下限是小于或等于每个值的最大整数。	FLOOR(m1) FLOOR(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
IF	IF 表达式	TS	通过将 IF 与比较运算符结合使用，可以从时间序列中筛选出数据点或创建由多个已整理的时间序列组成的混合时间序列。有关更多信息，请参阅 使用 IF 表达式。	有关示例，请参阅使用 IF 表达式。	✓
INSIGHT_RULE_METRIC	INSIGHULE_MIC(ruleNe, metricNe)	TS	使用 INSIGHT_RULE_METRIC 可在 Contributor Insights 中从规则提取统计数据。有关更多信息，请参阅 绘制规则生成的指标的图表。		
LAMBDA	LAMBDmbdaFuionNam[, optionalarg]*)	TS TS[]	调用 Lambda 函数以查询源自非 CloudWatch 数据来源的指标。有关更多信息，请参阅 如何将参数传递给您的 Lambda 函数。		
LOG	TS TS[]	TS TS[]	时间序列的 LOG 返回的是该时间序列中每个值的自然对数值。	LOG(METRICS())	✓
LOG10	TS TS[]	TS TS[]	时间序列的 LOG10 返回的是该时间序列中每个值的以 10 为底的对数值。	LOG10(m1)	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
MAX	TS TS[]	S TS	单个时间序列的 MAX 将返回一个标量，表示指标中所有数据点的最大值。 如果输入时间序列数组，则 MAX 函数会创建并返回一个时间序列，其中包含用作输入的时间序列中每个数据点的最大值。 **Note** 如果您希望函数返回一个标量，则建议您不要在 CloudWatch 警报中使用此函数。例如，`MAX(m2)` 每当警报评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	MAX(m1)/m1 MAX(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
METRIC_COUNT	TS[]	S	返回时间序列数组中的指标数量。	m1/METRIC_COUNT(METRICS())	✓
METRICS	null 字符串	TS[]	METRICS() 函数返回的是请求中的所有 CloudWatch 指标。数学表达式不包括在内。 您可以在一个更大的表达式中使用 METRICS()，从而最终生成单个时间序列或一组时间序列。 您可以将 METRICS() 函数与一个字符串一起使用，仅返回在其 Id 字段中包含该字符串的绘成图表的指标。例如，表达式 SUM(METRICS("errors")) 一个是所有在其 Id 字段中具有“errors”的绘成图表的指标值的总和的时间序列。您还可以使用 SUM([METRICS(“4xx”), METRICS(“5xx”)]) 来匹配多个字符串。	AVG(METRICS()) SUM(METRICS("errors"))	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
MIN	TS TS[]	S TS	单个时间序列的 MIN 将返回一个标量，表示指标中所有数据点的最小值。 如果输入时间序列数组，则 MIN 函数会创建并返回一个时间序列，其中包含用作输入的时间序列中每个数据点的最小值。 如果输入时间序列数组，则 MIN 函数会创建并返回一个时间序列，其中包含用作输入的时间序列中每个数据点的最小值。 **Note** 如果您希望函数返回一个标量，则建议您不要在 CloudWatch 警报中使用此函数。例如，`MIN(m2)` 每当警报评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定	m1-MIN(m1) MIN(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
			的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。		

函数	参数	返回类型*	描述	示例	支持跨账户？

MINUTE
 	
HOUR
 	
DAY
 	
DATE
 	
MONTH
 	
YEAR
 	
EPOCH
	TS	TS	这些函数采用时间序列的周期和范围，并返回一个新的非稀疏时间序列，其中每个值都基于其时间戳。
			
			• MINUTE 返回 0 到 59 之间的整数非稀疏时间序列，表示原始时间序列中每个时间戳的 UTC 分钟。
			• HOUR 返回 0 到 23 之间的整数非稀疏时间序列，表示原始时间序列中每个时间戳的 UTC 小时。
			• DAY 返回 1 到 7 之间的整数非稀疏时间序列，表示原始时间序列中每个时间戳的 UTC 星期。1 表示星期一，7 表示星期日。
			• DATE 返回 1 到 31 之间的整数非稀疏时间序列，表示原始时间序列中每个时间戳的 UTC 日期。
			• MONTH 返回 1 到 12 之间的整数非稀疏时间序列，表示原始时间序列中每个时间戳
				MINUTE(m1)
				
				IF(DAY(m1)<6,m1) 仅返回工作日（即星期一到星期五）的指标。
				
				IF(MONTH(m1) == 4,m1) 仅返回 4 月发布的指标。
					✓

函数	参数	返回类型*	描述	示例	支持跨账户？
			的 UTC 月份。1 表示 1 月，12 表示 12 月。 • YEAR 返回一个整数非稀疏时间序列，表示原始时间序列中每个时间戳的 UTC 年份。 • EPOCH 返回一个整数非稀疏时间序列，表示自原始时间序列中每个时间戳的纪元以来的 UTC 时间（以秒为单位）。纪元为 1970 年 1 月 1 日。		
PERIOD	TS	S	以秒为单位返回度量的时间段。有效输入是指标，而不是其他表达式的结果。	m1/PERIOD(m1)	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
RATE	TS TS[]	TS TS[]	返回指标每秒的变化率。此项的计算结果是最新数据点值和上一个数据点值之差除以两个值的秒数之差。 **Important** 对表达式（此类表达式将 RATE 函数用于具有稀疏数据的指标）设置警报可能会产生无法预测的行为，因为评估告警时获取的数据点范围可能会因上次发布数据点的时间而异。	RATE(m1) RATE(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
REMOVE_EMPTY	TS[]	TS[]	从时间序列数组中删除所有不带数据点的时间序列。最终将获得一个时间序列数组，其中每个时间序列包含至少一个数据点。 **Note** 我们建议您不要在 CloudWatch 告警中使用此函数。每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	REMOVE_EMPTY(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
RUNNING_SUM	TS TS[]	TS TS[]	返回一个时间序列，其中包含原始时间序列中值的运行总和。 **Note** 我们建议您不要在 CloudWatch 告警中使用此函数。每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	RUNNING_SUM([m1,m2])	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
SEARCH	搜索表达式	一个或多个 TS	返回一个或多个与您指定的搜索条件匹配的时间系列。SEARCH 函数使您能够使用一个表达式将多个相关时间序列添加到图表中。该图表会动态更新，以包含稍后添加并与搜索条件匹配的新指标。有关更多信息，请参阅 在图表中使用搜索表达式。 您无法根据 SEARCH 表达式创建告警。原因在于搜索表达式会返回多个时间序列，而基于数学表达式的告警只能监视一个时间序列。 如果您登录 CloudWatch 跨账户可观测性中的监控账户，则 SEARCH（搜索）函数将在源账户和该监控账户中查找指标。		✓

函数	参数	返回类型*	描述	示例	支持跨账户？
SERVICE_QUOTA	作为使用指标的 TS	TS	返回给定使用指标的服务配额。您可以使用此选项来可视化当前用量与配额的比较情况，并设置在接近配额时向您发送警告的警报。有关更多信息，请参阅 AWS 使用情况指标。		✓
SLICE	(TS[], S, S) 或 (TS[], S)	TS[] TS	检索时间序列数组的一部分。这在与 SORT 结合使用时特别有用。例如，可以从时间序列数组中排除顶部结果。 可以使用两个标量参数来定义要返回的时间序列集。这两个标量将定义要返回的数组的开始（包含）和结束（不包含）。该数组从零开始建立索引，因此，数组中的第一个时间序列是时间序列 0。或者，可以仅指定一个值，CloudWatch 将返回以该值开头的所有时间序列。	SLICE(SORT(METRICS(), SUM, DESC), 0, 10) 从请求中的指标数组返回具有最高 SUM 值的 10 个指标。 SLICE(SORT(METRICS(), AVG, ASC), 5) 按 AVG 统计数据对指标数组进行排序，然后返回所有时间序列（具有最低 AVG 的 5 个时间序列除外）。	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
SORT	(TS[], FUNCT SORT_(R) (TS[], FUNCT SORT_(R, S)	TS[]	根据您指定的函数，对时间序列数组进行排序。您使用的函数可以是 AVG、MIN、MAX 或 SUM。排序顺序可以是 ASC（升序，最低值排第一位）或 DESC（降序，最高值排第一位）。您可以选择在排序顺序后指定一个数字来充当限制。例如，如果指定的限制为 5，则仅返回排序中的前 5 个时间序列。 当图表上显示此数学函数时，图表中每个指标的标签也将进行排序和编号。	SORT(METRICS(), AVG, DESC, 10) 计算每个时间序列的平均值，对时间序列进行排序（开头为最高值），并仅返回具有最高平均值的 10 个时间序列。 SORT(METRICS(), MAX, ASC) 根据 MAX 统计数据对指标数组进行排序，然后以升序顺序返回所有指标。	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
STDDEV	TS TS[]	S TS	单个时间序列的 STDDEV 将返回一个标量，表示指标中所有数据点的标准偏差。一组时间序列的 STDDEV 将返回单个时间序列。 **Note** 如果您希望函数返回一个标量，则建议您不要在 CloudWatch 警报中使用此函数。例如，STDDEV(m2) 每当警报评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	m1/STDDEV(m1) STDDEV(METRICS())	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
SUM	TS TS[]	S TS	单个时间序列的 SUM 将返回一个标量，表示指标中所有数据点的值的总和。一组时间序列的 SUM 将返回单个时间序列。 **Note** 如果您希望函数返回一个标量，则建议您不要在 CloudWatch 警报中使用此函数。例如，SUM(m1)。每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于评估期指定的数量的数据点数。当请求了额外数据时，此函数的行为将有所不同。	SUM(METRICS())/SUM(m1) SUM([m1,m2]) SUM(METRICS("errors"))/SUM(METRICS("requests"))*100	✓

函数	参数	返回类型*	描述	示例	支持跨账户？
TIME_SERIES	S	TS	返回一个非稀疏时间序列，其中每个值都设置为标量参数。	TIME_SERIES(MAX(m1)) TIME_SERIES(5*AVG(m1)) TIME_SERIES(10)	✓

*使用仅返回标量数字的函数是无效的，因为表达式的所有最终结果都必须是单个时间序列或一组时间序列。而应将这些函数用作返回时间序列的较大表达式的一部分。

使用 IF 表达式

通过将 IF 与比较运算符结合使用，可以从时间序列中筛选出数据点或创建由多个已整理的时间序列组成的混合时间序列。

IF 使用以下参数：

```
IF(condition, trueValue, falseValue)
```

如果条件数据点的值为 0，则条件的计算结果为 FALSE；如果条件的值为任何其他值（无论该值是正值还是负值），则条件的计算结果为 TRUE。如果条件是一个时间序列，则会为每个时间戳单独计算该条件。

下面列出了有效语法。对于其中的每个语法，输出是单时间序列。

- IF(TS ***#####*** S, S | TS, S | TS)

  > **Note**
  >
  > 如果 `TS comparison operator S` 为 TRUE 但 `metric2` 没有对应的数据点，则输出将为 0。

- IF(TS, TS, TS)
- IF(TS, S, TS)

- IF(TS, TS, S)
- IF(TS, S, S)
- IF(S, TS, TS)

以下部分提供了这些语法的更多详细信息和示例。

IF(TS ***#####*** S, scalar2 | metric2, scalar3 | metric3)

对应的输出时间序列值：

- 如果 TS ***#####*** S 为 TRUE，则值为 scalar2 或 metric2
- 如果 TS ***#####*** S 为 FALSE，则值为 scalar3 或 metric3
- 如果 TS ***#####***为 TRUE 且 metric2 中不存在对应的数据点，则值为 0。
- 如果 TS ***#####***为 FALSE 且 metric3 中不存在对应的数据点，则值为 0。
- 是一个空白时间序列（如果 metric3 中没有对应的数据点，或者如果表达式中省略了 scalar3/metric3）

IF(metric1, metric2, metric3)

对于 metric1 的每个数据点，对应的输出时间序列值：

- 值为 metric2（如果 metric1 的对应数据点为 TRUE）。
- 值为 metric3（如果 metric1 的对应数据点为 FALSE）。
- 值为 0（如果 metric1 的对应数据点为 TRUE 且 metric2 中没有对应的数据点）。
- 如果 metric1 的对应数据点为 FALSE 且 metric3 中没有对应的数据点，则将被删除
- 如果 metric1 的对应数据点为 FALSE 且表达式中忽略了 metric3，则将被删除
- 如果 metric1 对应的数据点缺失，则将被删除。

下表显示了此语法示例。

指标或函数	值
(metric1)	[1, 1, 0, 0, -]
(metric2)	[30, -, 0, 0, 30]

指标或函数	值
(metric3)	[0, 0, 20, -, 20]
IF(metric1, metric2, metric3)	[30, 0, 20, 0, -]

IF(metric1, scalar2, metric3)

对于 metric1 的每个数据点，对应的输出时间序列值：

- 值为 scalar2（如果 metric1 的对应数据点为 TRUE）。
- 值为 metric3（如果 metric1 的对应数据点为 FALSE）。
- 被删除（如果 metric1 的对应数据点为 FALSE 且 metric3 中没有对应的数据点，或者如果表达式中省略了 metric3）。

指标或函数	值
(metric1)	[1, 1, 0, 0, -]
scalar2	5
(metric3)	[0, 0, 20, -, 20]
IF(metric1, scalar2, metric3)	[5, 5, 20, -, -]

IF(metric1, metric2, scalar3)

对于 metric1 的每个数据点，对应的输出时间序列值：

- 值为 metric2（如果 metric1 的对应数据点为 TRUE）。
- 值为 scalar3（如果 metric1 的对应数据点为 FALSE）。
- 值为 0（如果 metric1 的对应数据点为 TRUE 且 metric2 中没有对应的数据点）。
- 如果 metric1 中的相应数据点不存在，则被删除。

指标或函数	值
(metric1)	[1, 1, 0, 0, -]
(metric2)	[30, -, 0, 0, 30]
scalar3	5
IF(metric1, metric2, scalar3)	[30, 0, 5, 5, -]

IF(scalar1, metric2, metric3)

对应的输出时间序列值：

- 值为 metric2（如果 scalar1 为 TRUE）。
- 值为 metric3（如果 scalar1 为 FALSE）。
- 是一个空白时间序列（如果表达式中省略了 metric3）。

IF 表达式的使用案例示例

以下示例说明 IF 函数的可能用法。

- 要仅显示指标的低值，请使用：

  IF(metric1<400, metric1)
- 要将指标中的每个数据点更改为两个值之一，以显示原始指标的相对高值和低值，请使用：

  IF(metric1<400, 10, 2)
- 要为延迟超过阈值的每个时间戳显示 1，并为所有其他数据点显示 0，请使用：

  IF(latency>threshold, 1, 0)

将指标数学与 GetMetricData API 操作配合使用

您可以使用 `GetMetricData` 来通过数学表达式执行计算，还可以在一个 API 调用中检索大量指标数据。有关更多信息，请参阅 GetMetricData。

指标数学异常检测

指标数学异常检测是一项功能，您可以使用该功能针对单个指标和指标数学表达式的输出创建异常检测告警。您可以使用这些表达式来创建能可视化异常检测范围的图表。该功能支持基本的算术函数、比较和逻辑运算符以及大多数其他函数。

指标数学异常检测不支持以下功能：

- 在同一行中包含多个 ANOMALY_DETECTION_BAND 的表达式。
- 包含 10 个以上指标或数学表达式的表达式。
- 包含 METRICS 表达式的表达式。
- 包含 SEARCH 函数的表达式。
- 使用 DP_PERF_INSIGHTS 函数的表达式。
- 以不同时段使用指标的表达式。
- 使用高精度指标作为输入的指标数学异常检测器。

有关此功能的更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 CloudWatch 异常检测。

在图表中使用搜索表达式

搜索表达式是一种类型的数学表达式，您可以将其添加到 CloudWatch 图表中。搜索表达式可让您快速向图表添加多个相关指标。它们还使您能够创建动态图表，从而自动将相应的指标添加到其显示中，即使您在第一次创建图表时不存在这些指标也是如此。

例如，您可以创建一个搜索表达式来显示区域中所有实例的 `AWS/EC2 CPUUtilization` 指标。如果您稍后启动新实例，则新实例的 `CPUUtilization` 会自动添加到图表中。

在图表中使用搜索表达式时，搜索会在指标名称、命名空间、维度名称和维度值中查找搜索表达式。您可以使用布尔运算符进行更复杂、更强大的搜索。搜索表达式只能找到在过去两周内报告了数据的指标。

您无法根据 SEARCH 表达式创建告警。这是因为搜索表达式可返回多个时间序列，而基于数学表达式的告警只能监视一个时间序列。

如果您使用的是 CloudWatch 跨账户可观测性中的监控账户，则您的搜索表达式可以在与该监控账户关联的源账户中查找指标。

主题

- CloudWatch 搜索表达式语法
- CloudWatch 搜索表达式示例
- 使用搜索表达式创建 CloudWatch 图表

CloudWatch 搜索表达式语法

有效的搜索表达式具有以下格式。

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic')
```

例如：

```
SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')
```

- 查询中单词 SEARCH 之后的第一部分（用大括号括起来）是要搜索的指标架构。指标架构包含指标命名空间以及一个或多个维度名称。在搜索查询中包含指标架构是可选的。如果指定，则指标架构必须包含命名空间，并且可以（可选）包含在该命名空间中有效的一个或多个维度名称。

  除非命名空间或维度名称包含空格或非字母数字字符，否则不需要在指标架构中使用引号。在这种情况下，您必须用双引号将包含这些字符的名称引起来。
- SearchTerm 也是可选的，但有效的搜索必须包含指标架构和/或 SearchTerm。SearchTerm 通常包含一个或多个账户 ID、指标名称或维度值。SearchTerm 可以通过部分匹配和精确匹配来包含要搜索的多个词语。它还可以包含布尔运算符。

  在 SearchTerm 中使用账户 ID 仅适用于设置为 CloudWatch 跨账户可观测性监控账户的账户。SearchTerm 中账户 ID 的语法是 `:aws.AccountId = 444455556666`。您也可以使用 `'LOCAL'` 来指定监控账户本身：`:aws.AccountId = 'LOCAL'`

  有关更多信息，请参阅 CloudWatch 跨账户可观测性。

  SearchTerm 可以包含一个或多个指示符，例如 MetricName=（如本例中所示），但不需要使用指示符。

  指标架构和 SearchTerm 必须一起包含在一对单引号中。
- Statistic 是任何有效 CloudWatch 统计数据的名称。它必须用单引号引起来。有关更多信息，请参阅 统计数据。

上述示例在 AWS/EC2 命名空间中搜索任何具有 InstanceId 作为维度名称的指标。它会返回找到的所有 CPUUtilization 指标，图表显示 Average 统计数据。

搜索表达式只能找到在过去两周内报告了数据的指标。

搜索表达式限制

最大搜索表达式查询大小为 1024 个字符。一个图表上最多可以包含 100 个搜索表达式。一个图表可以显示多达 500 个时间序列。

CloudWatch 搜索表达式：令牌化

当您指定 SearchTerm 时，搜索函数会搜索令牌，即 CloudWatch 自动根据完整的指标名称、维度名称、维度值和命名空间生成的子字符串。CloudWatch 生成的令牌在原始字符串中以驼峰式大小写区分。数字字符也可用作新令牌的开头，非字母数字字符用作分隔符，从而在非字母数字字符之前和之后创建令牌。

相同类型的令牌分隔符字符的连续字符串会生成一个令牌。

所有生成的令牌都是小写的。下表显示了一些生成的令牌示例。

原始字符串	生成的令牌
CustomCount1	customcount1 ，custom, count, 1
SDBFailure	sdbfailure ，sdb, failure
Project2-trial333	project2trial333 ，project, 2, trial, 333

CloudWatch 搜索表达式：部分匹配项

当您指定 SearchTerm 时，搜索词也会被令牌化。CloudWatch 根据部分匹配项查找指标，部分匹配项是根据搜索词生成的单个令牌与根据指标名称、命名空间、维度名称或维度值生成的单个令牌的匹配项。

与单个令牌匹配的部分匹配搜索是不区分大小写的。例如，使用以下任一搜索词可能会返回 CustomCount1 指标：

- count

- Count
- COUNT

但是，使用 couNT 作为搜索词不会找到 couNT，因为搜索词 CustomCount1 中的大写会被令牌化为 cou 和 NT。

搜索还可以匹配复合令牌，复合令牌是在原始名称中连续出现的多个令牌。为与复合令牌匹配，搜索是区分大小写的。例如，如果原始词是 CustomCount1，则 CustomCount 或 Count1 的搜索成功，但 customcount 或 count1 的搜索不成功。

CloudWatch 搜索表达式：精确匹配项

您可以定义搜索，以便通过使用双引号将搜索词中需要精确匹配的部分引起来，仅查找搜索词的精确匹配项。这些双引号被括在将整个搜索词引起来的单引号中。例如，**SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum')** 找到确切的字符串 CustomCount1（如果它在名为 MyNamespace 的命名空间中作为指标名称、维度名称或维度值存在）。但是，搜索 **SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum')** 或 **SEARCH(' {MyNamespace}, "Custom" ', 'Maximum')** 找不到该字符串。

您可以在单个搜索表达式中组合部分匹配词和精确匹配词。例如，**SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')** 会返回名为 ConsumedLCUs 的 Elastic Load Balancing 指标和包含令牌 flow 的所有 Elastic Load Balancing 指标或维度。

使用精确匹配也是查找具有特殊字符（例如非字母数字字符或空格）的名称的好方法，如下例所示。

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ',
 'Maximum')
```

CloudWatch 搜索表达式：排除指标架构

到目前为止显示的所有示例都包括指标架构（在大括号中）。省略指标架构的搜索也是有效的。

例如，**SEARCH(' "CPUUtilization" ', 'Average')** 返回所有与字符串 CPUUtilization 精确匹配的指标名称、维度名称、维度值和命名空间。在 AWS 指标命名空间中，这可以包括多个服务（包括 Amazon EC2、Amazon ECS、SageMaker 等）中的指标。

要将此搜索范围缩小到只有一个 AWS 服务，最佳做法是在指标架构中指定命名空间以及任何必要的维度，如以下示例所示。虽然这会缩小对 AWS/EC2 命名空间的搜索范围，但如果您已将 CPUUtilization 定义为其他指标的维度值，它仍会返回这些指标的结果。

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average')
```

或者，您可以在 SearchTerm 中添加命名空间，如以下示例所示。但是在本示例中，搜索将与任何 AWS/EC2 字符串匹配，即使它是自定义维度名称或值。

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average')
```

CloudWatch 搜索表达式：在搜索中指定属性名称

以下搜索 "CustomCount1" 的精确匹配将返回具有该名称的所有指标。

```
SEARCH(' "CustomCount1" ', 'Maximum')
```

但它也返回维度名称、维度值或命名空间为 CustomCount1 的指标。要进一步构建搜索，您可以指定要在搜索中查找的对象类型的属性名称。以下示例搜索所有命名空间并返回名为 CustomCount1 的指标。

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum')
```

您还可以使用命名空间和维度名称/值对作为属性名称，如以下示例所示。这些示例中的第一个还演示了您也可以将属性名称与部分匹配搜索一起使用。

```
SEARCH(' InstanceType=micro ', 'Average')
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average')
```

CloudWatch 搜索表达式：非字母数字字符

非字母数字字符用作分隔符，并标记要将指标名称、维度名称、命名空间名称和搜索词分隔成令牌的位置。当词语被令牌化时，非字母数字字符会被剥离掉，不会出现在令牌中。例如，Network-Errors_2 生成令牌 network、errors 和 2。

您的搜索词可以包含任何非字母数字字符。如果这些字符出现在搜索词中，则可以在部分匹配中指定复合令牌。例如，以下所有搜索都将查找名为 Network-Errors-2 或 NetworkErrors2 的指标。

```
network/errors
network+errors
network-errors
```

```
Network_Errors
```

当您进行精确值搜索时，精确搜索中使用的任何非字母数字字符都必须是出现在要搜索的字符串中的正确字符。例如，如果您要查找 Network-Errors-2，则 "Network-Errors-2" 的搜索成功，但 "Network_Errors_2" 搜索不成功。

执行精确匹配搜索时，必须使用反斜杠转义以下字符。

```
" \ ( )
```

例如，要按精确匹配查找指标名称 Europe\France Traffic(Network)，请使用搜索词 **"Europe \\France Traffic\(Network\)"**

CloudWatch 搜索表达式：布尔运算符

搜索支持在 SearchTerm 中使用布尔运算符 AND、OR 和 NOT。布尔运算符被括在用于将整个搜索词引起来的单引号中。布尔运算符区分大小写，因此 and、or 和 not 作为布尔运算符无效。

您可以在搜索中显式使用 AND，例如 **SEARCH('{AWS/EC2,InstanceId} network AND packets', 'Average')**。在搜索词之间不使用任何布尔运算符会隐式地搜索它们，就好像存在 AND 运算符，因此 **SEARCH(' {AWS/EC2,InstanceId} network packets ', 'Average')** 产生相同的搜索结果。

使用 NOT 从结果中排除数据子集。例如，**SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average')** 返回您的所有实例（i-1234567890123456 实例除外）的 CPUUtilization。您还可以使用 NOT 子句作为唯一的搜索词。例如，**SEARCH('NOT Namespace=AWS ', 'Maximum')** 产生所有自定义指标（具有不包括 AWS 的命名空间的指标）。

您可以在一个查询中使用多个 NOT 短语。例如，**SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average')** 返回区域中的所有实例（但维度值为 ProjectA 或 ProjectB 的实例除外）的 CPUUtilization。

您可以组合布尔运算符以进行更强大、更详细的搜索，如以下示例所示。使用括号对运算符进行分组。

接下来的两个示例都返回包含 EC2 和 EBS 命名空间中的 ReadOps 的所有指标名称。

```
SEARCH(' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum')
```

```
SEARCH(' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum')
```

以下示例将以前的搜索范围缩小为仅包含 ProjectA 的结果，这可能是维度的值。

```
SEARCH(' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum')
```

以下示例使用嵌套分组。它返回所有函数的 Errors 的 Lambda 指标，以及名称中包含字符串 ProjectA 或 ProjectB 的函数的 Invocations。

```
SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

CloudWatch 搜索表达式：使用数学表达式

您可以在图表中的数学表达式内使用搜索表达式。

例如，**SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum'))** 返回您的所有 Lambda 函数的 Errors 指标的总和。

对搜索表达式和数学表达式使用单独的行可能会产生更有用的结果。例如，假设您在图表中使用以下两个表达式。第一行显示您的各个 Lambda 函数的单独 Errors 行。该表达式的 ID 为 e1。第二行添加另一行，以显示所有函数的错误总和。

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum')
SUM(e1)
```

CloudWatch 搜索表达式示例

以下示例演示了更多搜索表达式用法和语法。让我们首先在区域内的所有实例中搜索 CPUUtilization，然后查看变体。

此示例为区域中的每个实例显示一行，从而显示 AWS/EC2 命名空间中的 CPUUtilization 指标。

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average')
```

将 InstanceId 更改为 InstanceType 会更改图表，以便为区域中使用的每个实例类型显示一行。来自每个类型的所有实例的数据针对该实例类型聚合为一行。

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average')
```

以下示例按实例类型聚合 CPUUtilization，并为包含字符串 micro 的每个实例类型显示一行。

```
SEARCH('{AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',
 'Average')
```

此示例缩小了上一个示例，将 InstanceType 更改为 t2.micro 实例的精确搜索。

```
SEARCH('{AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',
 'Average')
```

以下搜索删除了查询的 {metric schema} 部分，因此所有命名空间中的 CPUUtilization 指标都显示在图表中。这可以返回不少结果，因为该图表为来自每个 AWS 服务的 CPUUtilization 指标包含多个行（沿不同维度聚合）。

```
SEARCH('MetricName="CPUUtilization" ', 'Average')
```

要稍微缩小这些结果范围，可以指定两个特定的指标名称空间。

```
SEARCH('MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average')
```

前面的示例是使用一个搜索查询来搜索特定的多个名称空间的唯一方法，因为您在每个查询中只能指定一个指标架构。但是，要添加更多结构，可以在图表中使用两个查询，如以下示例所示。在此示例中，还通过指定用于聚合 Amazon ECS 数据的维度来添加更多结构。

```
SEARCH('{AWS/ECS ClusterName}, MetricName="CPUUtilization" ', 'Average')
SEARCH(' {AWS/EBS} MetricName="CPUUtilization" ', 'Average')
```

以下示例返回名为 ConsumedLCUs 的 Elastic Load Balancing 指标和包含令牌 flow 的所有 Elastic Load Balancing 指标或维度。

```
SEARCH('{AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')
```

以下示例使用嵌套分组。它返回所有函数的 Errors 的 Lambda 指标，以及名称中包含字符串 ProjectA 或 ProjectB 的函数的 Invocations。

```
SEARCH('{AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

以下示例显示了所有自定义指标，不包括 AWS 服务生成的指标。

```
SEARCH('NOT Namespace=AWS ', 'Average')
```

以下示例显示具有包含字符串 Errors 作为其名称一部分的指标名称、命名空间、维度名称和维度值的指标。

```
SEARCH('Errors', 'Average')
```

以下示例将搜索缩小到精确匹配。例如，此搜索会查找指标名称 Errors，但不会查找名为 ConnectionErrors 或 errors 的指标。

```
SEARCH(' "Errors" ', 'Average')
```

以下示例说明如何在搜索词的指标架构部分中指定包含空格或特殊字符的名称。

```
SEARCH('{"Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum')
```

CloudWatch 跨账户可观测性搜索表达式示例

CloudWatch 跨账户可观测性示例

如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则可以使用 SEARCH（搜索）函数从指定的源账户返回指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

以下示例从账户 ID 为 111122223333 的账户中检索所有 Lambda 指标。

```
SEARCH(' AWS/Lambda :aws.AccountId = 111122223333 ', 'Average')
```

以下示例从两个账户中检索所有 AWS/EC2 指标：111122223333 和 777788889999。

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 777788889999) ', 'Average')
```

以下示例从源账户 111122223333 和监控账户本身检索所有 AWS/EC2 指标。

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 'LOCAL') ', 'Average')
```

以下示例将检索账户 444455556666 中指标 MetaDataToken 的 SUM，维度为 InstanceId。

```
SEARCH('{AWS/EC2,InstanceId} :aws.AccountId=444455556666 MetricName=\"MetadataNoToken
\"','Sum')
```

使用搜索表达式创建 CloudWatch 图表

在 CloudWatch 控制台上，在将图表添加到控制面板时或通过使用 Metrics（指标）视图，可以访问搜索功能。

您无法根据 SEARCH 表达式创建告警。原因在于搜索表达式会返回多个时间序列，而基于数学表达式的告警只能监视一个时间序列。

将具有搜索表达式的图表添加到现有控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Dashboards，然后选择一个控制面板。
3. 选择 Add widget。
4. 选择折线或堆积面积，然后选择配置。
5. 在 Graphed metrics（已绘制图表指标）选项卡上，选择 Add a math expression (添加数学表达式)。
6. 对于 Details (详细信息)，输入所需的搜索表达式。例如，**`SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')`**
7. （可选）要向图表中添加其他搜索表达式或数学表达式，请选择 Add a math expression (添加数学表达式)
8. （可选）在添加搜索表达式后，您可以为每个指标指定在图表图例上显示的动态标签。动态标签显示有关指标的统计数据，并在刷新控制面板或图表时自动进行更新。要添加动态标签，请选择Graphed metrics（绘制的指标），然后选择动态标签。

   默认情况下，添加到标签的动态值显示在标签开头。然后，您可以单击指标的标签值以编辑标签。有关更多信息，请参阅 使用动态标签。
9. （可选）要向图表中添加单个指标，请选择 All metrics (所有指标) 选项卡并向下钻取到所需的指标。
10. (可选) 要更改图表上显示的时间范围，请选择图表顶部的 custom (自定义)，或 custom (自定义) 左侧的一个时间段。
11. （可选）横向注释帮助控制面板用户快速查看指标的峰值何时达到特定级别，或者指标是否在预定义的范围内。要添加横向注释，请选择 Graph options (图表选项)，然后选择 Add horizontal annotation (添加横向注释)：

    a. 对于 Label (标签)，输入注释的标签。

    b. 对于 Value (值)，输入显示横向注释的指标值。

c. 对于 Fill，指定是否将填充阴影用于此注释。例如，为要填充的相应区域选择 `Above` 或 `Below`。如果您指定 `Between`，则另一个 `Value` 字段将出现，并且将填充两个值之间的图表区域。

d. 对于 Axis，指定 `Value` 中的数字是否在图表包含多个指标的情况下表示与左侧 Y 轴或右侧 Y 轴关联的指标。

您可以通过在注释的左列中选择颜色方块来更改注释的填充色。

重复这些步骤可向同一个图表添加多个横向注释。

要隐藏注释，请清除该注释的左列中的复选框。

要删除注释，请在 Actions 列中选择 x。

12. （可选）垂直注释可帮助您标记图表中的里程碑，例如操作事件或部署的开始和结束。要添加垂直注释，请选择 Graph options (图表选项)，然后选择 Add vertical annotation (添加垂直注释)：

a. 对于 Label (标签)，输入注释的标签。要仅在注释中显示日期和时间，请将 Label (标签) 字段留空。

b. 对于 Date (日期)，指定垂直注释出现的日期和时间。

c. 对于 Fill (填充)，指定是否在垂直注释前面或后面使用填充阴影或者在两个垂直注释之间使用填充阴影。例如，为要填充的相应区域选择 `Before` 或 `After`。如果您指定 `Between`，则另一个 `Date` 字段将出现，并且将填充两个值之间的图表区域。

重复这些步骤可向同一个图表添加多个垂直注释。

要隐藏注释，请清除该注释的左列中的复选框。

要删除注释，请在 Actions 列中选择 x。

13. 选择 Create widget。

14. 选择 Save dashboard。

使用“Metrics”(指标) 视图绘制所搜索指标的图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择指标。

3. 在搜索字段中，输入要搜索的令牌：例如，**`cpuutilization t2.small`**。

将会显示与您的搜索匹配的结果。

4. 要绘制与搜索匹配的所有指标的图表，请选择 Graph search (图表搜索)。

   或者

   要细化您的搜索，请选择搜索结果中显示的一个命名空间。

5. 如果您选择了命名空间来缩小结果范围，则可以执行以下操作：

   a. 要为一个或多个指标绘制图表，请选中每个指标旁边的复选框。要选择所有指标，请选中表的标题行中的复选框。

   b. 要细化您的搜索，请将鼠标指针悬停在某个指标名称上，然后选择 Add to search（添加到搜索）或 Search for this only（仅为此搜索）。

   c. 要查看指标的帮助，请选择指标名称，然后选择 What is this?。

   所选指标会显示在图表上。

6. （可选）选择搜索栏中的一个按钮以编辑搜索词的该部分。

7. （可选）要将图表添加到控制面板，请选择 Actions (操作)，然后选择 Add to dashboard (添加到控制面板)。

获取指标的统计数据

CloudWatch 统计数据定义

统计数据是指定时间段内的指标数据汇总。当您绘制某个指标的图表或检索其统计数据时，您可以指定时间段（例如 5 分钟），用于计算每个统计值。例如，如果时间段为 5 分钟，则总数是在此五分钟内所收集的所有样本值的总数，而最小值是此五分钟内收集的最低值。

CloudWatch 支持指标的以下统计数据。

- 样本数是在该时间段内的数据点数。
- 总数是在该时间段内收集的所有数据点值的总数。
- 平均值是指定时间段内 `Sum/SampleCount` 的值。
- 最小值是指定时间段内的最小观察值。
- 最大值是指定时间段内的最大观察值。

- 百分位数 (p) 表示某个值在数据集中的相对位置。例如，p95 指第 95 个百分位数，表示该时间段内 95% 的数据低于此值，5% 的数据高于此值。百分位数可帮助您更好地了解指标数据的分布情况。
- 切尾均值 (TM) 是两个指定边界之间的所有值的均值。计算此均值时，两个边界以外的值均被忽略。您可以将边界定义为 0 到 100 之间的一个或两个数字，最多 10 位小数。这些数字可以是绝对值或百分比。例如，tm90 计算的是去掉 10% 具有最高值的数据点后的平均值。TM(2%:98%) 计算的是去掉 2% 最小数据点和 2% 最大数据点后的平均值。TM(150:1000) 计算的是去掉所有小于或等于 150 或大于 1000 的数据点后的平均值。
- 四分位均值 (IQM) 是四分位间距的切尾均值，或中间 50% 的值。它等同于 TM(25%:75%)。
- 缩尾均值 (WM) 类似于切尾均值。但是，计算缩尾均值时，不会忽略边界以外的值，而是将它们视为等于相应边界边缘的值。经过此规范化处理之后再计算平均值。您可以将边界定义为 0 到 100 之间的一个或两个数字，最多 10 位小数。例如，wm98 将计算平均值，同时将 2% 的最高值视为等于第 98 个百分位数的值。WM(10%:90%) 将计算平均值，同时将 10% 的最大数据点视为 90% 的边界值，并将 10% 的最小数据点视为 10% 的边界值。
- 百分等级 (PR) 是满足某一固定阈值的值的百分比。例如，PR(:300) 返回的是值等于或小于 300 的数据点的百分比。PR(100:2000) 返回的是值介于 100 到 2000 之间的数据点的百分比。

  百分位数排名的下限不含本数，上限包含本数。
- 切尾计数 (TC) 指所选切尾均值统计数据范围内的数据点数。例如，tc90 返回的是不包括处于 10% 最高值区间内的任何数据点的数据点数。TC(0.005:0.030) 返回的是值介于 0.005（不含）和 0.030（含）之间的数据点数。
- 切尾总数 (TS) 指所选切尾均值统计数据范围内数据点的值的总数。它等同于（切尾均值）*（切尾计数）。例如，ts90 返回的是不包括处于 10% 最高值区间内的任何数据点的数据点总数。TS(80%:) 返回的是不包括值处于 80% 最低值区间内的任何数据点的值的总数。

> **Note**
>
> 对于“切尾均值”、“切尾计数”、“切尾总数”和“缩尾均值”，如果将两个边界定义为固定值而不是百分比，则计算时将包括等于较高边界的值，但不包括等于较低边界的值。

语法

对于“切尾均值”、“切尾计数”、“切尾总数”和“缩尾均值”，以下语法规则适用：

- 使用括号和带有百分号的一个或两个数字来定义边界，以用作数据集中位于指定的两个百分位数之间的值。例如，TM(10%:90%) 将仅使用位于第 10 和第 90 个百分位数之间的值。TM(:95%) 将使用处于数据集最低值区间到第 95 个百分位数之间的值，忽略 5% 具有最高值的数据点。
- 使用括号和无百分号的一个或两个数字来定义边界，以用作数据集中位于指定的确切值之间的值。例如，TC(80:500) 将仅使用介于 80（不含）和 500（含）之间的值。TC(:0.5) 将仅使用等于或小于 0.5 的值。
- 使用一个数字（无括号）以百分比形式来计算，计算过程忽略高于指定百分位数的数据点。例如，tm99 将计算均值，同时忽略 1% 具有最高值的数据点。其与 TM(:99%) 的计算结果相同。
- 在指定范围时，切尾均值、切尾计数、切尾总数和缩尾均值均可以使用大写字母缩写表示，例如 TM(5%:95%)、TM(100:200) 或 TM(:95%)。仅指定一个数字时，只能使用小写字母缩写，例如 tm99。

统计数据使用案例

- 切尾均值对于样本规模较大的指标（例如网页延迟）最实用。例如，tm99 会忽略可能由网络问题或人为错误导致的极高异常值，以便为典型请求的平均延迟提供更准确的数据。同样，TM(10%:) 会忽略 10% 的最低延迟值，例如由缓存命中导致的延迟值。此外，TM(10%:99%) 将排除这两种类型的异常值。我们建议您使用切尾均值来监测延迟。
- 每当您使用切尾均值时，最好随时关注切尾计数，以确保在切尾均值计算中使用的值的数量足以具有统计意义。
- 百分等级使您能够将值归入范围的“条柱”中，而且您可以使用它来手动创建直方图。为此，请将值归入各种不同的条柱，例如 PR(:1)、PR(1:5)、PR(5:10) 和 PR(10:)。以条形图的方式可视化呈现其中每个条柱，这样就得到一个直方图。

  百分位数排名的下限不含本数，上限包含本数。

百分位数与切尾均值

百分位数（如 p99）和切尾均值（如 tm99）衡量的是相似但并不完全相同的值。p99 和 tm99 二者都会忽略 1% 具有最高值的数据点，这些数据点被视为异常值。在此之后，p99 为剩余 99% 值的最大值，而 tm99 为剩余 99% 的值的平均值。如果您正在研究 Web 请求的延迟，p99 告诉您的是最糟糕的客户体验，而 tm99 告诉您的是平均客户体验，二者都会忽略异常值。

如果您正寻求优化客户体验，那么切尾均值是适用于监测的理想延迟统计数据。

对使用百分位数、切尾均值和其他一些统计数据的要求

CloudWatch 需要原始数据点来计算以下统计数据：

- 百分位数
- 切尾均值
- 四分位均值
- 缩尾均值
- 切尾总数
- 切尾计数
- 百分等级

如果您不使用原始数据而是改用统计数据集发布数据，只有满足以下条件之一，才能检索此数据的这些类型统计数据：

- 统计数据集的 SampleCount 值为 1，且最小值、最大值和总和均相等。
- 最小值和最大值相等，总和等于最小值乘以 SampleCount。

以下 AWS 服务包含支持这些类型统计数据的指标。

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

此外，当任何指标值为负数时，指标的这些类型统计数据将不可用。

以下示例说明如何获取资源（例如 EC2 实例）的 CloudWatch 指标的统计数据。

示例

- 获取特定资源的统计数据
- 跨资源聚合统计数据
- 按 Auto Scaling 组聚合统计数据

- 按亚马逊机器映像（AMI）聚合统计数据

获取特定资源的统计数据

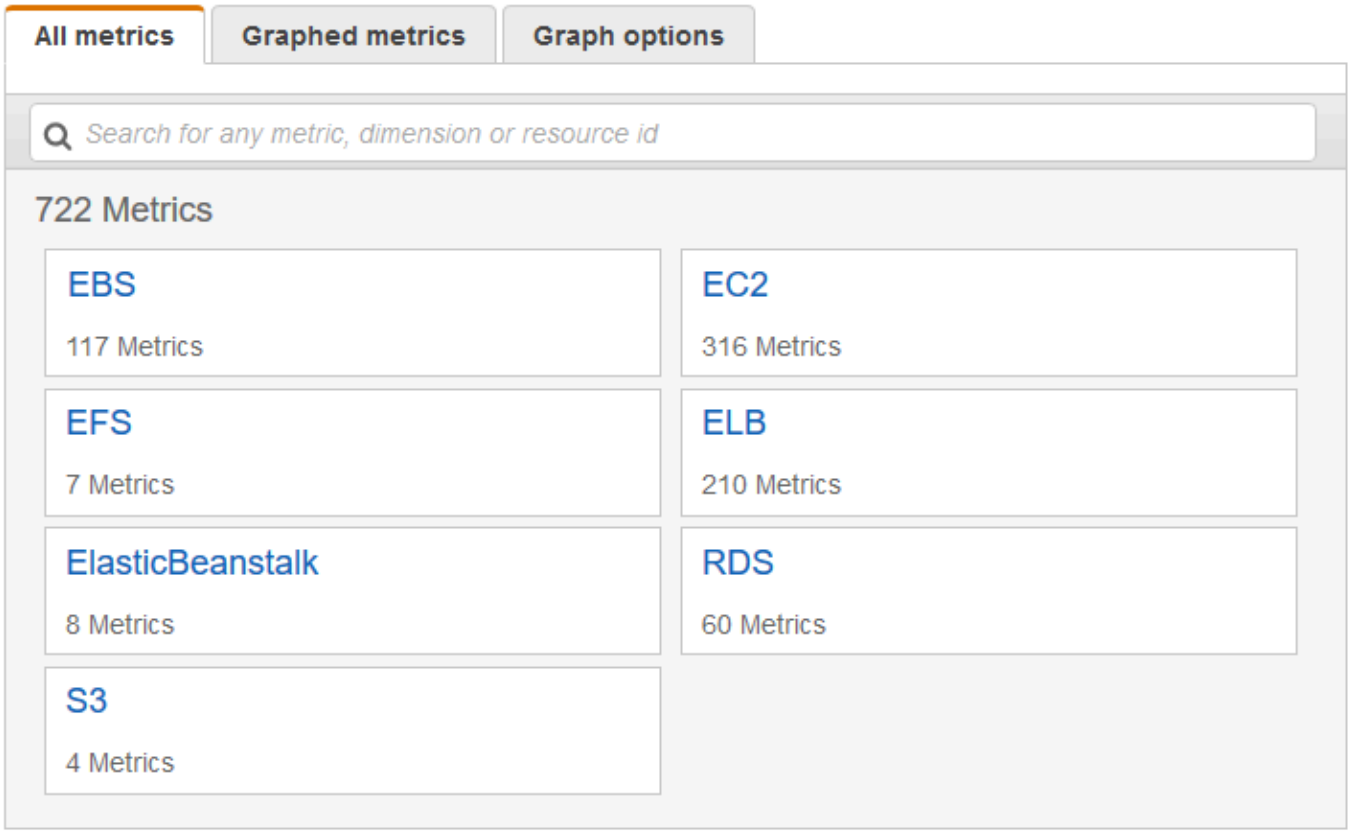
以下示例说明如何确定特定 EC2 实例的最大 CPU 使用率。

要求

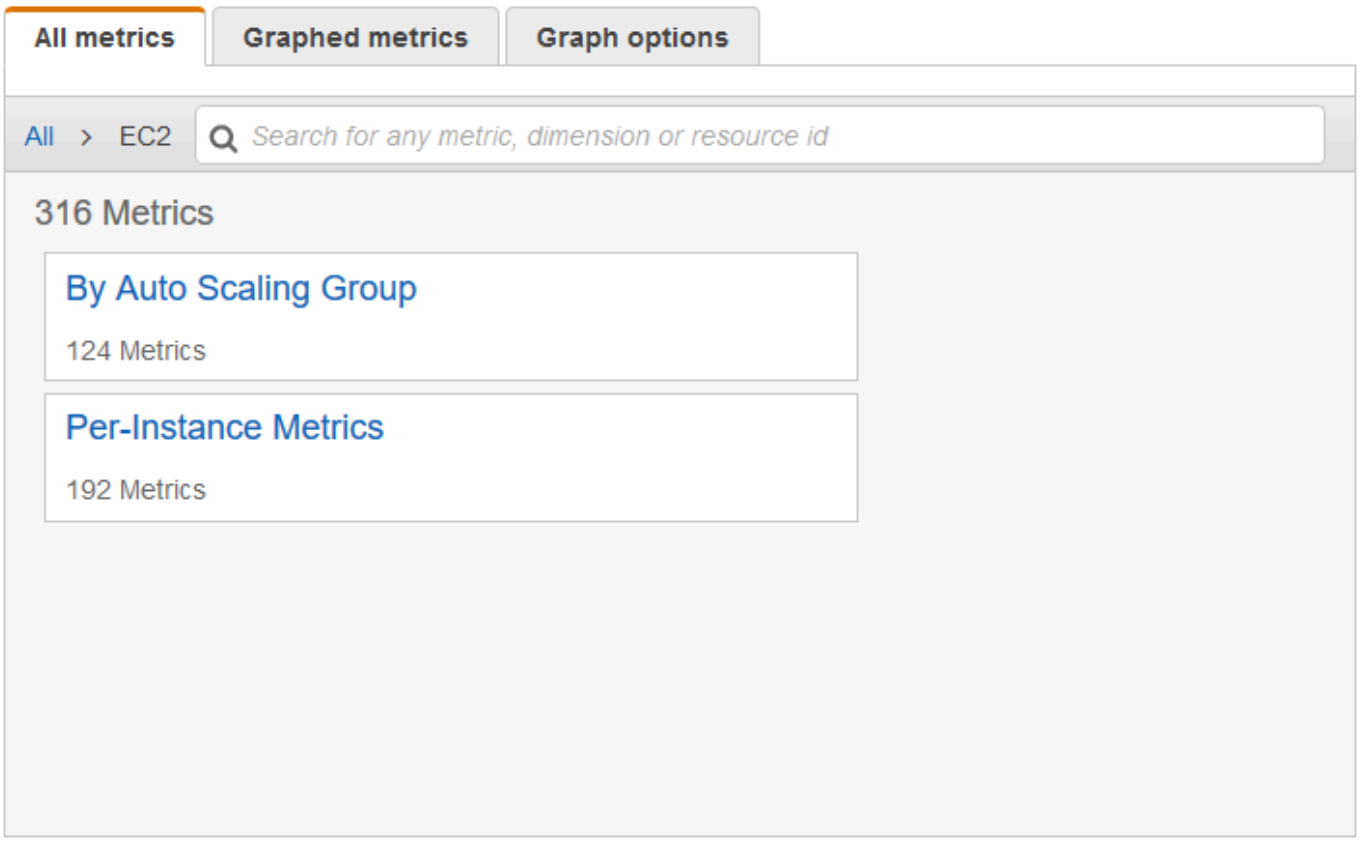
- 您必须拥有实例的 ID。可使用 Amazon EC2 控制台或 describe-instances 命令获取实例 ID。
- 默认情况下，基本监控已启用，但您可以启用详细监控。有关更多信息，请参阅《Amazon EC2 用户指南》中的对实例启用或禁用详细监控。

使用控制台显示特定实例的平均 CPU 使用率

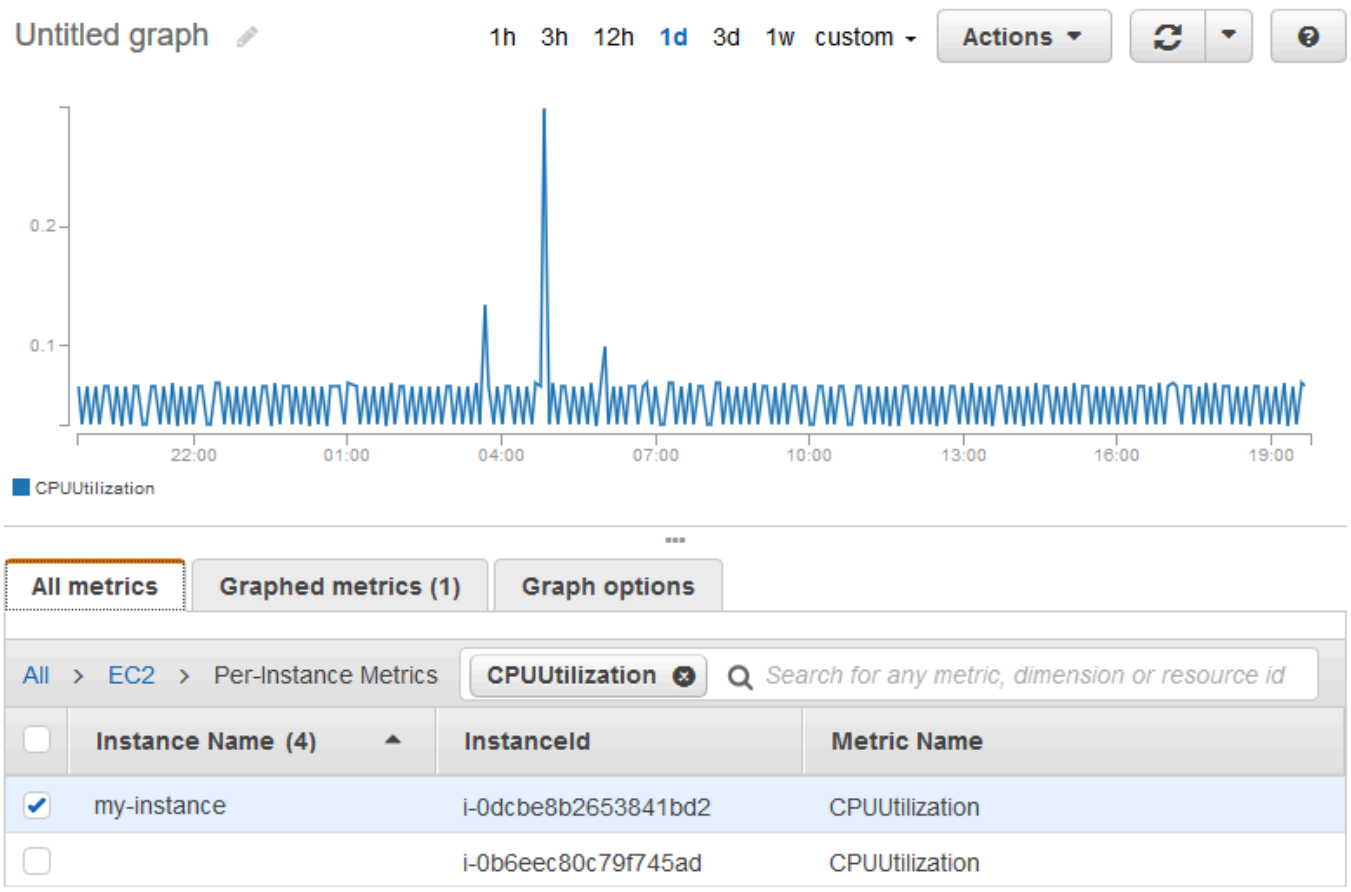
1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择 EC2 指标命名空间。



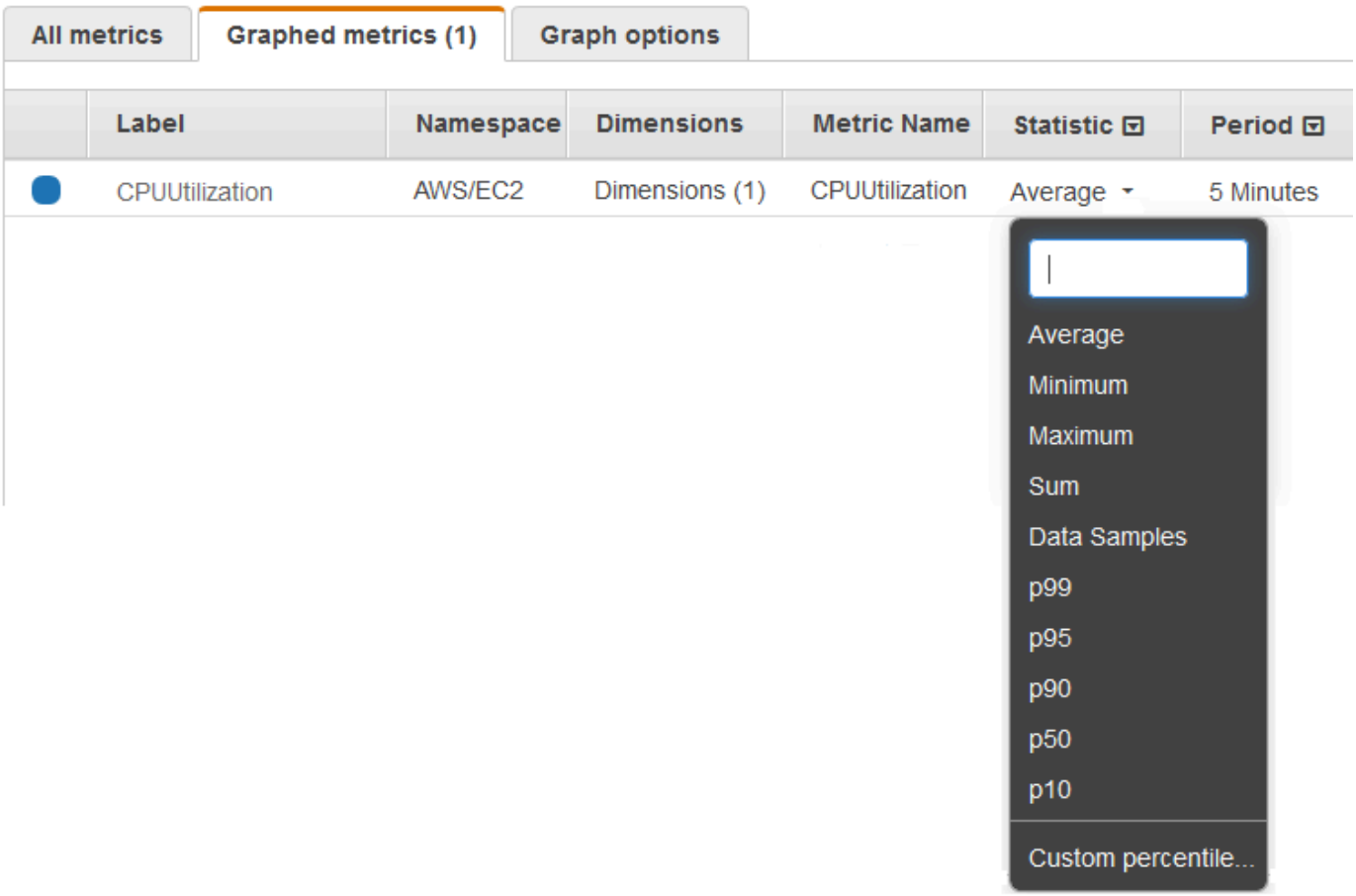
4. 选择 Per-Instance Metrics（每个实例的指标）维度。



5. 在搜索框中，输入 **`CPUUtilization`** 并按 Enter。选择特定实例的行，这将显示该实例的 `CPUUtilization` 指标的图表。要更改图表的名称，请选择铅笔图标。要更改时间范围，请选择某个预定义的值或选择 custom。



6. 要更改统计数据，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择某个统计数据或预定义百分位数，或指定自定义百分位数（例如 **p99.999**）。



7. 要更改时间段，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择其他值。

使用 AWS CLI 获取每个 EC2 实例的 CPU 使用率

使用 get-metric-statistics 命令（如下所示）可获取指定实例的 CPUUtilization 指标：

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Maximum \
--start-time 2016-10-18T23:18:00 --end-time 2016-10-19T23:18:00 --period 360
```

返回的统计数据是以请求的 24 小时时间为间隔的 6 分钟数值。每个值表示特定的六分钟时间段内的指定实例的最大 CPU 使用率百分比。数据点不是按时间顺序返回的。下面显示了示例输出的开头（完整输出包括 24 小时内每 6 分钟时间段的数据点）：

```
{
    "Datapoints": [
        {
```

```
            "Timestamp": "2016-10-19T00:18:00Z",
            "Maximum": 0.33000000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T03:18:00Z",
            "Maximum": 99.670000000000002,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-19T07:18:00Z",
            "Maximum": 0.34000000000000002,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

跨资源聚合统计数据

您可以跨多个资源聚合 AWS 资源的指标。指标在区域之间完全独立，但您可以使用指标数学来跨区域聚合和转换类似指标。有关更多信息，请参阅 使用指标数学。

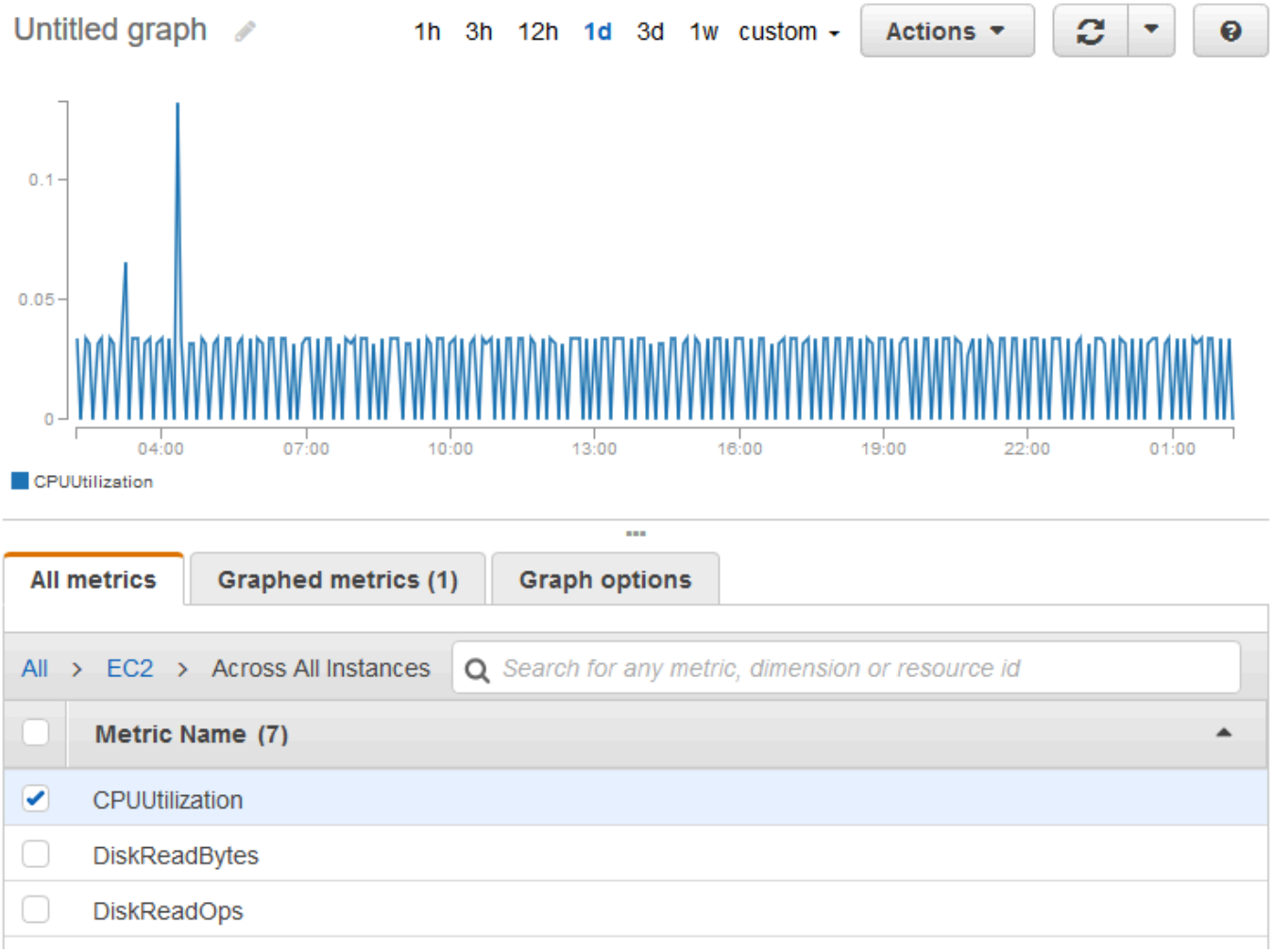
例如，您可以聚合已启用详细监控的 EC2 实例的统计数据。不包含使用基本监控的实例。因此，您必须启用详细监控 (另外收费)，这将提供以 1 分钟为间隔的数据。有关更多信息，请参阅《Amazon EC2 用户指南》中的对实例启用或禁用详细监控。

此示例说明如何获取 EC2 实例的平均 CPU 使用率。因为未指定任何维度，所以 CloudWatch 会返回 AWS/EC2 命名空间中所有维度的统计数据。要获取其他指标的统计信息，请参阅 发布 CloudWatch 指标的 AWS 服务。

> **Important**
>
> 此方法可以在 AWS 命名空间中检索所有维度，但不适用于发布到 CloudWatch 的自定义命名空间。对于自定义命名空间，必须指定与任意给定数据关联的完整的维度组，以检索包含数据点的统计数据。

显示 EC2 实例的平均 CPU 使用率

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择 EC2 命名空间，然后选择 Across All Instances。
4. 选择包含 `CPUUtilization` 的行，这将显示所有 EC2 实例的指标的图表。要更改图表的名称，请选择铅笔图标。要更改时间范围，请选择某个预定义的值或选择 custom。



5. 要更改统计数据，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择某个统计数据或预定义百分位数，或指定自定义百分位数（例如 **`p95.45`**）。
6. 要更改时间段，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择其他值。

使用 AWS CLI 获取 EC2 实例的平均 CPU 使用率

按以下所示使用 get-metric-statistics 命令：

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization
 --statistics "Average" "SampleCount" \
--start-time 2016-10-11T23:18:00 --end-time 2016-10-12T23:18:00 --period 3600
```

下面是示例输出：

```
{
    "Datapoints": [
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-12T07:18:00Z",
            "Average": 0.038235294117647062,
            "Unit": "Percent"
        },
        {
            "SampleCount": 240.0,
            "Timestamp": "2016-10-12T09:18:00Z",
            "Average": 0.16670833333333332,
            "Unit": "Percent"
        },
        {
            "SampleCount": 238.0,
            "Timestamp": "2016-10-11T23:18:00Z",
            "Average": 0.041596638655462197,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

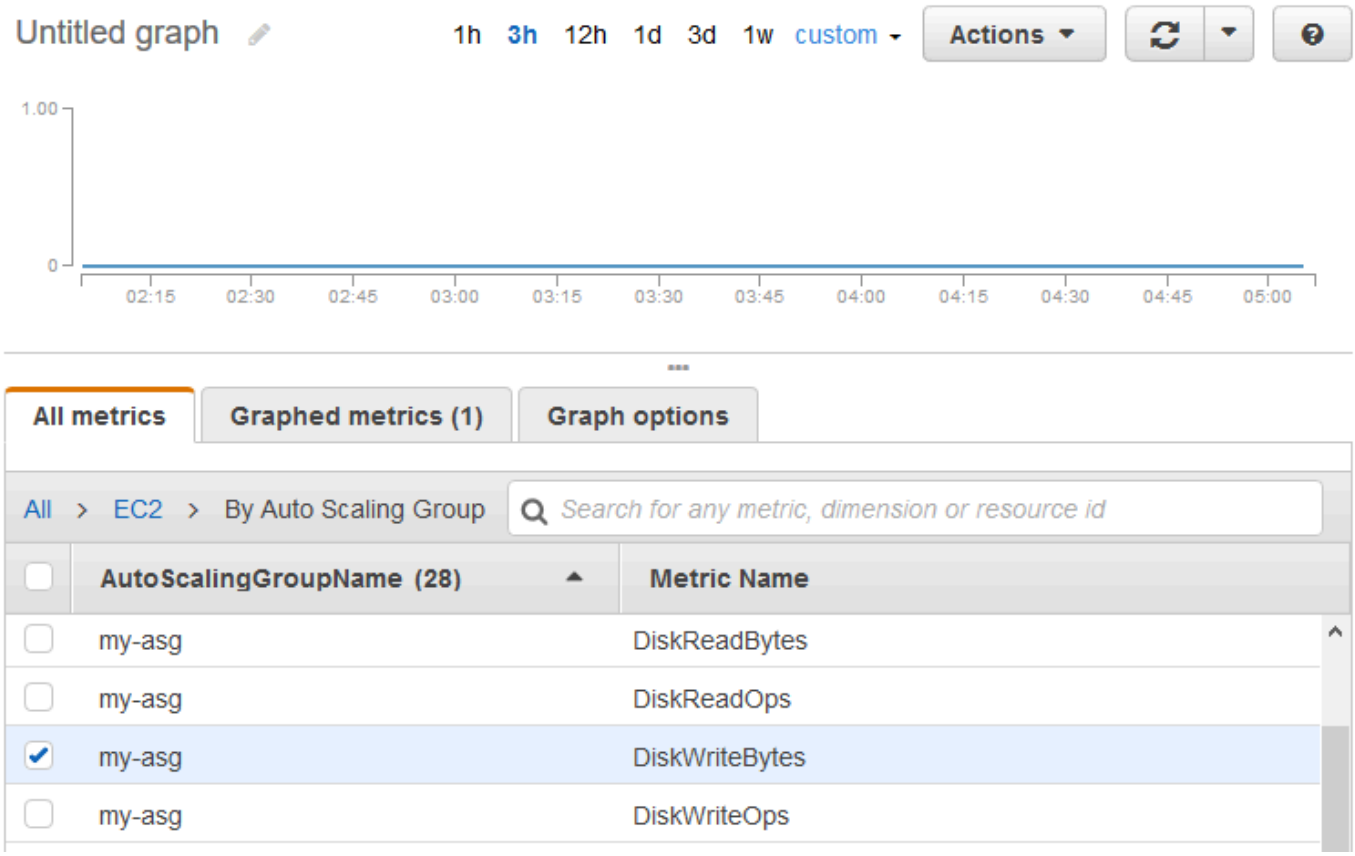
按 Auto Scaling 组聚合统计数据

您可以聚合 Auto Scaling 组中 EC2 实例的统计数据。指标在区域之间完全独立，但您可以使用 CloudWatch 指标数学来聚合和转换来自多个区域的指标。您还可以使用跨账户控制面板对来自不同账户的指标执行指标数学。

此示例说明如何获取为一个 Auto Scaling 组写入磁盘的字节总数。总数以 1 分钟为周期、24 小时为间隔针对指定 Auto Scaling 组中的所有 EC2 实例计算得出。

使用控制台显示 Auto Scaling 组中实例的 DiskWriteBytes

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择 EC2 命名空间，然后选择 By Auto Scaling Group。
4. 选择 DiskWriteBytes 指标和特定 Auto Scaling 组所在的行，随后将显示该 Auto Scaling 组中实例的指标的图表。要更改图表的名称，请选择铅笔图标。要更改时间范围，请选择某个预定义的值或选择 custom。



5. 要更改统计数据，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择某个统计数据或预定义百分位数，或指定自定义百分位数（例如 **p95.45**）。
6. 要更改时间段，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择其他值。

使用 AWS CLI 获取 Auto Scaling 组中实例的 DiskWriteBytes

使用 get-metric-statistics 命令，如下所示。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name DiskWriteBytes
```

```
--dimensions Name=AutoScalingGroupName,Value=my-asg --statistics "Sum" "SampleCount" \
--start-time 2016-10-16T23:18:00 --end-time 2016-10-18T23:18:00 --period 360
```

下面是示例输出。

```
{
    "Datapoints": [
        {
            "SampleCount": 18.0,
            "Timestamp": "2016-10-19T21:36:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        },
        {
            "SampleCount": 5.0,
            "Timestamp": "2016-10-19T21:42:00Z",
            "Sum": 0.0,
            "Unit": "Bytes"
        }
    ],
    "Label": "DiskWriteBytes"
}
```

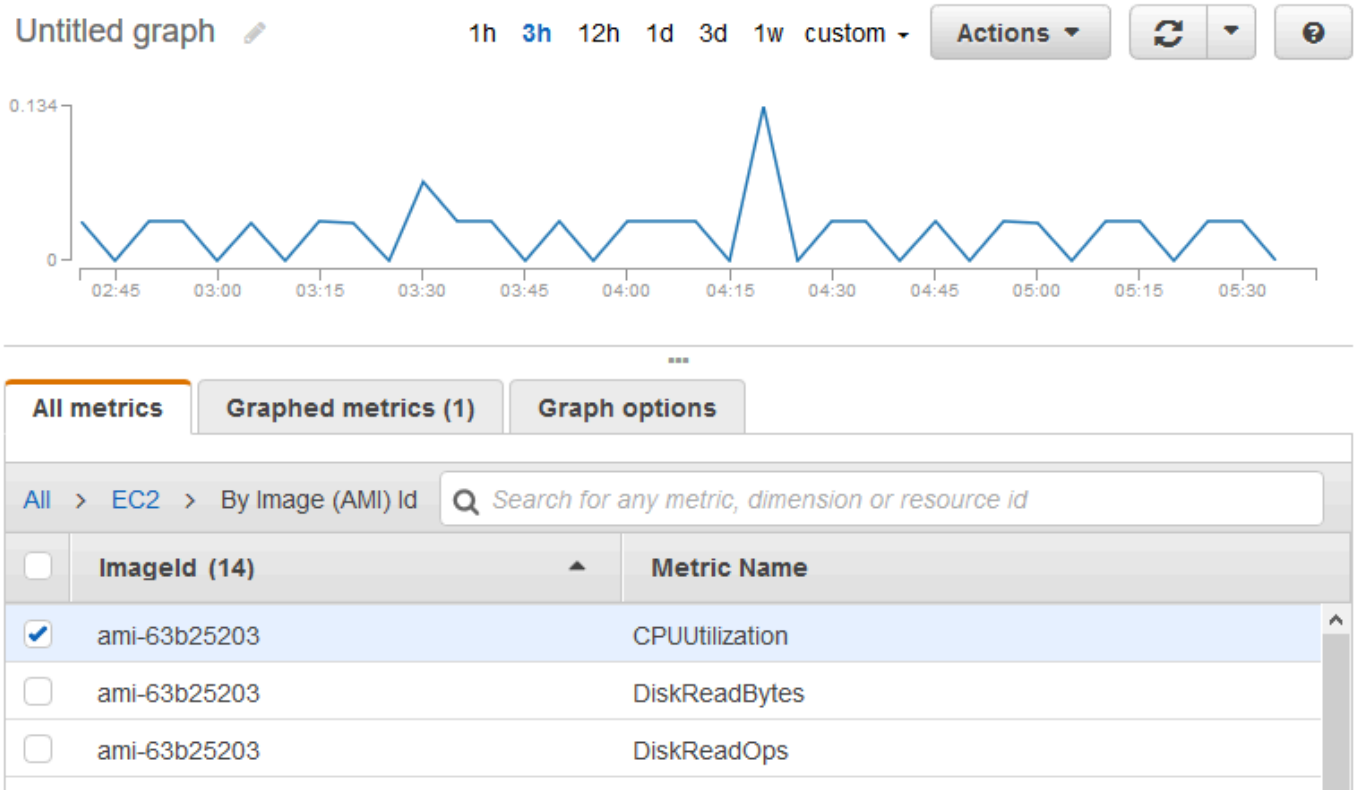
按亚马逊机器映像（AMI）聚合统计数据

您可以聚合已启用详细监控的 EC2 实例的统计数据。不包含使用基本监控的实例。有关更多信息，请参阅《Amazon EC2 用户指南》中的对实例启用或禁用详细监控。

此示例说明如何确定使用指定 AMI 的所有实例的平均 CPU 使用率。平均值以 60 秒为时间间隔 1 天为周期。

使用控制台按 AMI 显示平均 CPU 使用率

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 选择 EC2 命名空间，然后选择 By Image (AMI) Id。
4. 选择 CPUUtilization 指标和特定 AMI 的行，这将显示指定 AMI 的指标的图表。要更改图表的名称，请选择铅笔图标。要更改时间范围，请选择某个预定义的值或选择 custom。



5. 要更改统计数据，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择某个统计数据或预定义百分位数，或指定自定义百分位数（例如 **p95.45**）。
6. 要更改时间段，请选择 Graphed metrics（已绘制图表指标）选项卡。选择列标题或单个值，然后选择其他值。

使用 AWS CLI 获取按 AMI 聚合的平均 CPU 使用率

使用 get-metric-statistics 命令，如下所示。

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=ImageId,Value=ami-3c47a355 --statistics Average \
--start-time 2016-10-10T00:00:00 --end-time 2016-10-11T00:00:00 --period 3600
```

操作返回的统计信息是以 1 天时间为间隔的 1 分钟数值。每个数值代表运行指定 AMI 的 EC2 实例的平均 CPU 使用率百分比。下面是示例输出。

```
{
    "Datapoints": [
        {
            "Timestamp": "2016-10-10T07:00:00Z",
```

```
            "Average": 0.041000000000000009,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T14:00:00Z",
            "Average": 0.079579831932773085,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2016-10-10T06:00:00Z",
            "Average": 0.036000000000000011,
            "Unit": "Percent"
        },
        ...
    ],
    "Label": "CPUUtilization"
}
```

发布 自定义指标

您可使用 AWS CLI 或 API 将自己的指标发送到 CloudWatch。您可使用 AWS Management Console 查看所发布指标的统计信息图表。

CloudWatch 会将关于指标的数据储存为一系列数据点。每个数据点包含一个相关联的时间戳。您甚至可以发布被称为统计数据集 的聚合数据点集。

主题

- 高精度指标
- 使用维度
- 发布单一数据点
- 发布统计数据集
- 发布零值
- 停止发布指标

高精度指标

每个指标均为以下类型之一：

- 标准精度，数据粒度为一分钟
- 高精度，数据粒度为一秒

AWS 服务生成的指标在默认情况下为标准精度。在发布自定义指标时，您可以将其定义为标准精度或高精度。发布高精度指标时，CloudWatch 使用 1 秒的精度来存储指标，您可以按照 1 秒、5 秒、10 秒、30 秒或 60 秒的任意倍数的时间段读取和检索。

高精度指标让您对应用程序的亚分钟级活动有着更详细的直观认识。请记住，每次对自定义指标的 PutMetricData 调用都会收取费用，因此对高精度指标频繁调用 PutMetricData 会导致较高的费用。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

如果对高精度指标设置告警，您可以指定 10 秒或 30 秒时间段的高精度告警，也可以设置 60 秒的任意倍数时间段的定期告警。10 秒或 30 秒时间段的高精度警报会产生较高的费用。

使用维度

在自定义指标中，--dimensions 参数很常见。维度进一步阐明了指标是什么以及指标存储什么数据。您可以将最多 30 个维度分配给一个指标，每个维度通过一个名称和值对进行定义。

当使用不同的命令时，指定维度的方式也不同。使用 put-metric-data 时，将每个维度指定为 *MyName*=*MyValue* 格式，使用 get-metric-statistics 或 put-metric-alarm 时，将使用格式 Name=*MyName*、Value=*MyValue*。例如，以下命令将发布一个包含两个维度（分别名为 InstanceId 和 InstanceType）的 Buffers 指标。

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit
 Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

此命令检索同一指标的统计数据。使用逗号分隔单一维度的名称部分和值部分，但如果有多个维度，应在不同维度之间使用空格分隔。

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-
time 2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period
 60
```

如果单个指标包含多个维度，则在使用 get-metric-statistics 时必须为每个已定义的维度指定一个值。例如，Amazon S3 指标 BucketSizeBytes 包含维度 BucketName 和 StorageType，因此必须使用 get-metric-statistics 指定这两个维度。

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time
 2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace
 AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=MyBucketName
 Name=StorageType,Value=StandardStorage --output table
```

要查看为指标定义的维度，请使用 list-metrics 命令。

发布单一数据点

要发布新指标或现有指标的单一数据点，请使用带有一个值和时间戳的 put-metric-data 命令。例如，下列每项操作会发布一个数据点。

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 5 --timestamp 2016-10-20T12:00:02.000Z
```

如果使用新的指标名称调用此命令，CloudWatch 会为您创建一个指标。否则，CloudWatch 会将您的数据与您指定的现有指标进行关联。

> **Note**
>
> 当您创建指标时，可能需要 2 分钟时间，然后才能使用 get-metric-statistics 命令检索新指标的统计数据。但是，这可能需要最多 15 分钟时间，然后新指标才会出现在使用 list-metrics 命令检索的指标列表中。

尽管可以以千分之一秒粒度的时间戳发布数据点，但是 CloudWatch 会将数据聚合到一秒钟的最低粒度。CloudWatch 记录每个时间段内收到的值的平均值（所有项之和除以项数）以及同一个时间段内的样本数、最大值和最小值。例如，之前示例中的 PageViewCount 指标包含三个以若干秒为分隔的时间戳的数据点。如果将时间段设置为 1 分钟，则 CloudWatch 会聚合三个数据点，因为它们都具有 1 分钟时间段内的时间戳。

您可以使用 get-metric-statistics 命令，基于您发布的数据点来检索统计数据。

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount
 \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
```

```
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

下面是示例输出。

```
{
    "Datapoints": [
        {
            "SampleCount": 3.0,
            "Timestamp": "2016-10-20T12:00:00Z",
            "Average": 3.6666666666666665,
            "Maximum": 5.0,
            "Minimum": 2.0,
            "Sum": 11.0,
            "Unit": "None"
        }
    ],
    "Label": "PageViewCount"
}
```

发布统计数据集

您可在将数据发布到 CloudWatch 之前对数据进行聚合。如果您每分钟拥有多个数据点，则数据聚合会将调用 put-metric-data 的数目减到最少。例如，与其将彼此相隔 3 秒之内的三个数据点调用 put-metric-data 多次，不如使用 `--statistic-values` 参数将数据聚合到以一次调用发布的统计信息集中：

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService
 --statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch 需要原始数据点来计算百分位数。如果您改用统计数据集发布数据，则无法检索此数据的百分位数统计数据，除非满足以下条件之一：

- 统计数据集的 SampleCount 为 1。
- 统计数据集的 Minimum 和 Maximum 相等

发布零值

如果数据比较分散并有一些包含无关联数据的时间段，则可以选择为该时间段发布值零 (0) 或者无任何值。如果您通过定期调用 PutMetricData 来监控应用程序的运行状况，则可能需要发布零值来代替

无值。例如，可以设置一个当应用程序发布指标失败时每隔五分钟向您发送通知的 CloudWatch 告警。您想让这样的应用程序为不含任何关联数据的时间段发布零值。

如果想要追踪数据点的总数或需要最小和平均等统计数据包含 0 值数据点，也可以发布零值。

停止发布指标

要停止向 CloudWatch 发布自定义指标，请更改应用程序或服务的代码以停止使用 PutMetricData。CloudWatch 不会从应用程序中提取指标，它只会接收推送的内容，因此要停止发布指标，您必须在源头停止这些指标。

使用 Amazon CloudWatch 告警

您可以在 Amazon CloudWatch 中创建指标和复合警报。

- 指标告警可监控单个 CloudWatch 指标，或基于 CloudWatch 指标监控数学表达式的结果。告警根据指标或表达式在多个时间段内相对于某阈值的值执行一项或多项操作。操作可以是向 Amazon SNS 主题发送通知、执行 Amazon EC2 操作或 Amazon EC2 Auto Scaling 操作，或在 Systems Manager 中创建 OpsItem 或事件。
- 复合告警包括一个规则表达式，该表达式考虑您已创建的其他告警的告警状态。只有当规则的所有条件都得到满足时，复合告警才会进入“ALARM（告警）”状态。在复合告警的规则表达式中指定的告警可以包括指标告警和其他复合告警。

  使用复合告警可以减少告警噪音。您可以创建多个指标告警，还可以创建复合告警并仅为复合告警设置提示。例如，只有当所有底层指标告警都处于“ALARM（告警）”状态时，复合告警才可能进入“ALARM（告警）”状态。

  复合告警可以在改变状态时发送 Amazon SNS 通知，并且可以在进入“ALARM（告警）”状态时创建 Systems Manager OpsItems 或事件，但无法执行 EC2 操作或 Auto Scaling 操作。

> **Note**
>
> 您可以在您的 AWS 账户中创建任意数量的警报。

您可以为控制面板添加警报，以便监控与接收跨多个区域的 AWS 资源和应用程序的提醒。为控制面板添加警报以后，该警报会在处于 `INSUFFICIENT_DATA` 状态时变成灰色，在 `ALARM` 状态时变成红色。当处于 `OK` 状态时，警报没有颜色显示。

您还可以在 CloudWatch 控制台的导航面板中通过 Favorites and recents（收藏夹和最近记录）选项收藏最近访问过的警报。Favorites and recents（收藏夹和最近记录）选项中有对应的列，分别显示您收藏的警报和最近访问过的警报。

告警仅在告警状态更改时才会调用操作。使用 Auto Scaling 操作的告警除外。对于 Auto Scaling 操作，告警会在告警保持新状态时以每分钟一次的频率持续调用操作。

告警可以在同一账户中监视指标。如果您在 CloudWatch 控制台中启用了跨账户功能，则还可以创建告警，以监视其他 AWS 账户。不支持创建跨账户复合告警。支持创建使用数学表达式的跨账户告警，但跨账户告警不支持 `ANOMALY_DETECTION_BAND`、`INSIGHT_RULE` 和 `SERVICE_QUOTA` 函数。

> **Note**
>
> CloudWatch 不会测试或验证您指定的操作，也不会检测因试图调用不存在的操作而导致的任何 Amazon EC2 Auto Scaling 或 Amazon SNS 错误。请确保您的告警操作存在。

指标告警状态

指标告警可能具有以下几种状态：

- OK – 指标或表达式在定义的阈值范围内。
- ALARM – 指标或表达式超出定义的阈值。
- INSUFFICIENT_DATA（数据不足）– 告警刚刚启动，指标不可用，或者指标没有足够的数据以确定告警状态。

评估告警

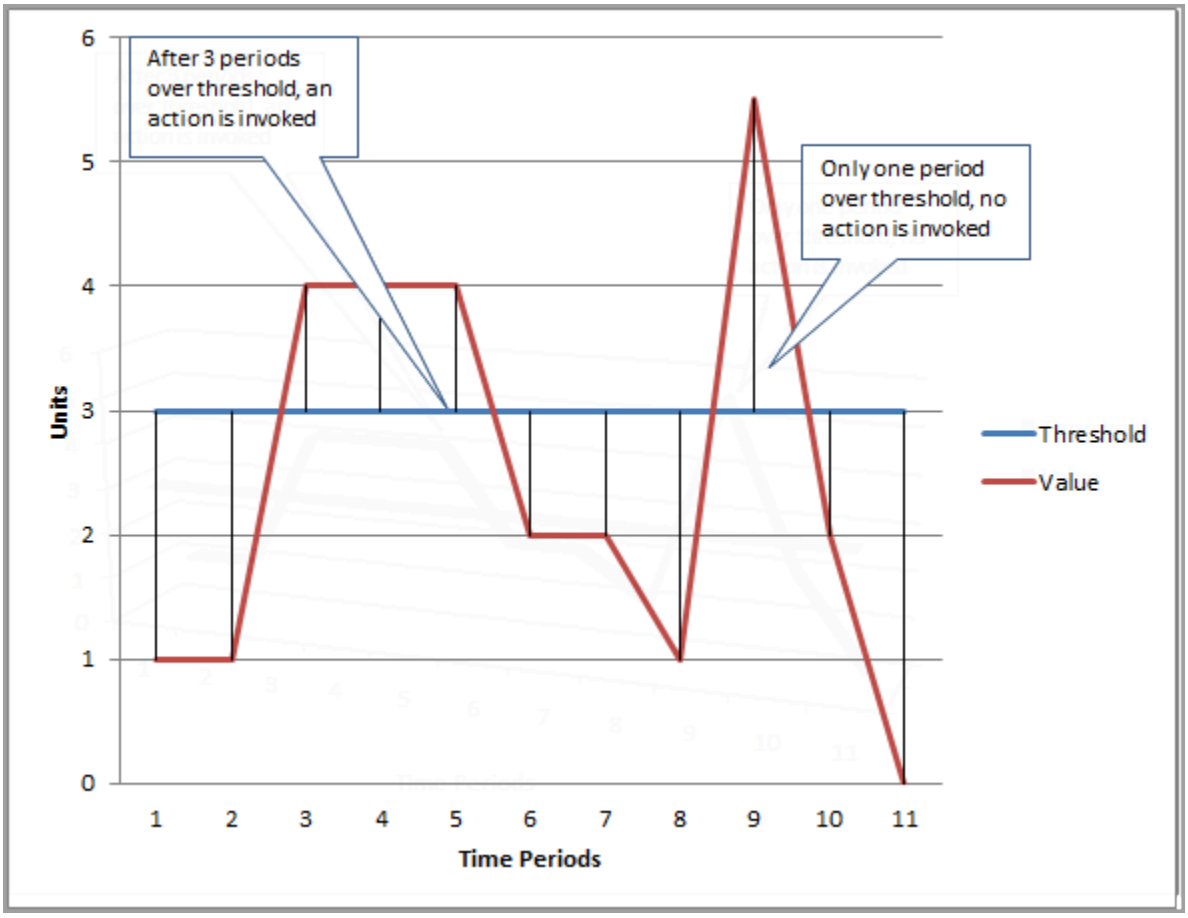
创建告警时，请指定三个设置，以启用 CloudWatch 评估在何时更改告警状态：

- 时间段是为了创建警报的各个数据点，而评估指标或表达式所用的时间长度。它以秒为单位。
- Evaluation Periods（评估期）是确定告警状态时要评估的最近时间段或数据点的数量。
- Datapoints to Alarm（触发告警的数据点数）是评估期内必须违例才能触发告警变为 ALARM（告警）状态的数据点数量。超出阈值的数据点不必是连续的，但它们必须全部在等于 Evaluation Period（评估期）的最近几个数据点范围内。

对于任何一分钟或更长的时间段，每分钟评估一次告警，评估基于周期和评估周期定义的时段。例如，如果周期为 5 分钟（300 秒），评估周期为 1，则在第 5 分钟结束时，告警将根据从 1 到 5 分钟的数据进行评估。然后，在第 6 分钟结束时，根据第 2 分钟到 6 分钟的数据对告警进行评估。

如果告警周期为 10 秒或 30 秒，则每 10 秒评估一次告警。

在下图中，指标告警的阈值设置为三个单位。Evaluation Periods（评估期）和 Datapoints to Alarm（触发告警的数据点数）均为 3。也就是说，在最近三个连续评估期中的所有现有数据点都高于阈值时，告警就会变为 ALARM（告警）状态。在该图中，在第三个到第五个时间段发生了这种情况。在第六个时间段，数值降至阈值以下，因此其中一个时间段被评估为未违例，且告警状态恢复为 OK（正常）。在第九个时间段，再次超出阈值，但仅一个时间段超出阈值。因此，告警状态保持为 OK（正常）。



在将 Evaluation Periods（评估期）和 Datapoints to Alarm（触发告警的数据点数）配置为不同的值时，您将设置“M（最大为 N）”告警。Datapoints to Alarm（触发告警的数据点数）为（“M”），而 Evaluation Periods（评估期）为（“N”）。评估间隔等于评估期数量乘以评估期时长。例如，如果为 1 分钟的评估期配置 4 个数据点（最大为 5 个数据点），则评估间隔为 5 分钟。如果为 10 分钟的评估期配置 3 个数据点（最大为 3 个数据点），则评估间隔为 30 分钟。

> **Note**
>
> 如果创建告警后不久便有数据点缺失，并且该指标在您创建告警之前便已报告给 CloudWatch，则 CloudWatch 在评估告警时会检索从创建告警之前算起的最近数据点。

告警操作

您可以指定告警在“OK（正常）”、“ALARM（告警）”和“INSUFFICIENT_DATA（数据不足）”状态之间更改状态时所执行的操作。

要过渡到这三种状态中的每一种，大多数操作都可以设置。除自动扩缩操作外，这些操作仅在状态转换时会发生，如果该情况持续数小时或数天，则不会再次执行。您可以利用允许告警进行多次操作这一事实，在超过阈值时发送一封电子邮件，然后在超出条件结束时发送另一封电子邮件。这可以帮助您验证扩展或恢复操作是否在预期时间触发，并按预期工作。

以下是受支持的警报操作。

- 通过使用 Amazon Simple Notification Service 主题通知一个或多个订阅用户。订阅用户既可以是应用程序，也可以是个人。有关 Amazon SNS 的更多信息，请参阅什么是 Amazon SNS？
- 调用 Lambda 函数。这是在警报状态变化时自动执行自定义操作的最简单方法。
- 基于 EC2 指标的警报还可以执行 EC2 操作，例如停止、终止、重启或恢复 EC2 实例。有关更多信息，请参阅 创建告警以停止、终止、重启或恢复 EC2 实例。
- 警报还可以执行操作来扩展自动扩缩组。有关更多信息，请参阅 Amazon EC2 Auto Scaling 的步进和简单扩展策略。
- 警报可以在 Systems Manager Ops Center 中创建 OpsItems，或在 AWS Systems Manager Incident Manager 中创建事件。只有在告警进入“ALARM（告警）”状态时才能执行这些操作。有关更多信息，请参阅 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

Lambda 警报操作

CloudWatch 警报可保证在给定状态变化时异步调用 Lambda 函数，但以下情况除外：

- 当该函数不存在时。
- 当 CloudWatch 未被授权调用该 Lambda 函数时。

如果 CloudWatch 无法访问 Lambda 服务或消息因其他原因被拒绝，CloudWatch 会不断重试，直到调用成功。Lambda 将消息加入队列并处理执行重试。有关这种执行模式的更多信息，包括有关 Lambda 如何处理错误的信息，请参阅《AWS Lambda 开发人员指南》中的 Asynchronous invocation。

可以调用同一账户中的 Lambda 函数，也可以调用其他 AWS 账户中的 Lambda 函数。

当您指定一个警报以调用 Lambda 函数作为警报操作时，您可以选择指定函数名称、函数别名或函数的特定版本。

当您将一个 Lambda 函数指定为警报操作时，必须为该函数创建一个资源策略，以允许 CloudWatch 服务主体调用该函数。

一种方法是使用 AWS CLI，如下例所示：

```
aws lambda add-permission \
--function-name my-function-name \
--statement-id AlarmAction \
--action 'lambda:InvokeFunction' \
--principal lambda.alarms.cloudwatch.amazonaws.com \
--source-account 111122223333 \
--source-arn arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name
```

或者，您可以创建一个类似于以下示例之一的策略，然后将其分配给该函数。

以下示例指定警报所在的账户，因此只有该账户（111122223333）中的警报才能调用该函数。

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [{
        "Sid": "AlarmAction",
        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.alarms.cloudwatch.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "111122223333"
            }
        }
    }]
}
```

以下示例的范围较窄，仅允许指定账户中的指定警报调用该函数。

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "AlarmAction",
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "lambda.alarms.cloudwatch.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
      }
    }]
}
```

我们不建议创建未指定来源账户的策略，因为此类策略容易受到混淆代理问题的影响。

从 CloudWatch 发送到 Lambda 的事件对象

当您将一个 Lambda 函数配置为警报操作时，CloudWatch 会在调用 Lambda 函数时向该函数传送一个 JSON 有效负载。此 JSON 有效负载用作函数的事件对象。您可以从此 JSON 对象中提取数据并将其用于您的函数。以下是指标警报中事件对象的示例。

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:444455556666:alarm:lambda-demo-metric-
alarm',
  'accountId': '444455556666',
  'time': '2023-08-04T12:36:15.490+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'lambda-demo-metric-alarm',
    'state': {
      'value': 'ALARM',
      'reason': 'test',
      'timestamp': '2023-08-04T12:36:15.490+0000'
    },
    'previousState': {
      'value': 'INSUFFICIENT_DATA',
      'reason': 'Insufficient Data: 5 datapoints were unknown.',
      'reasonData':
 '{"version":"1.0","queryDate":"2023-08-04T12:31:29.591+0000","statistic":"Average","period":60
[],"threshold":5.0,"evaluatedDatapoints":[{"timestamp":"2023-08-04T12:30:00.000+0000"},
{"timestamp":"2023-08-04T12:29:00.000+0000"},
{"timestamp":"2023-08-04T12:28:00.000+0000"},
```

```
{"timestamp":"2023-08-04T12:27:00.000+0000"},
{"timestamp":"2023-08-04T12:26:00.000+0000"}]}',
      'timestamp': '2023-08-04T12:31:29.595+0000'
    },
    'configuration': {
      'description': 'Metric Alarm to test Lambda actions',
      'metrics': [
        {
          'id': '1234e046-06f0-a3da-9534-EXAMPLEe4c',
          'metricStat': {
            'metric': {
              'namespace': 'AWS/Logs',
              'name': 'CallCount',
              'dimensions': {
                'InstanceId': 'i-12345678'
              }
            },
            'period': 60,
            'stat': 'Average',
            'unit': 'Percent'
          },
          'returnData': True
        }
      ]
    }
  }
}
```

以下是复合警报中事件对象的示例。

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:SuppressionDemo.Main',
  'accountId': '111122223333',
  'time': '2023-08-04T12:56:46.138+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'CompositeDemo.Main',
    'state': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
```

```
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:56:46.138+0000'
    },
    'previousState': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:54:46.138+0000',
      'actionsSuppressedBy': 'WaitPeriod',
      'actionsSuppressedReason': 'Actions suppressed by WaitPeriod'
    },
    'configuration': {
      'alarmRule': 'ALARM(CompositeDemo.FirstChild) OR
 ALARM(CompositeDemo.SecondChild)',
      'actionsSuppressor': 'CompositeDemo.ActionsSuppressor',
      'actionsSuppressorWaitPeriod': 120,
      'actionsSuppressorExtensionPeriod': 180
    }
  }
}
```

配置 CloudWatch 告警处理缺失数据的方式

有时，并非指标的每个预期数据点都会报告到 CloudWatch。例如，当连接中断、服务器出现故障或指标设计为仅间歇性地报告数据时，可能会发生这种情况。

您可以通过 CloudWatch 指定在评估告警时如何处理缺失的数据点。这可帮助您对告警进行配置，使其仅在适合所监控的数据类型时变为 ALARM（告警）状态。您可以避免在缺失数据没有指示问题时进行误报。

与每个告警始终处于三种状态之一类似，向 CloudWatch 报告的每个特定数据点将属于以下三个类别之一：

- 未违例（在阈值范围内）
- 违例（超出阈值）

- 缺失

对于每个告警，您可以指定 CloudWatch 将缺失数据点视为以下任一项：

- notBreaching – 将缺失数据点视为“良好”，并在阈值范围内
- breaching – 将缺失数据点视为“不良”，并超出阈值
- ignore（忽略）– 保持当前告警状态
- missing（缺失）– 如果告警评估范围内的所有数据点都缺失，则告警将转变为“INSUFFICIENT_DATA（数据不足）”状态。

最佳选择取决于指标的类型和警报的用途。例如，如果您使用持续报告数据的指标创建应用程序回滚警报，您可能需要将缺失数据点视为超出阈值，因为它可能表示出错了。但对于仅在错误发生时生成数据点的指标（如 Amazon DynamoDB 中的 ThrottledRequests），应将缺失数据视为 notBreaching。默认行为是 missing。

> **Important**
>
> 如果缺失指标数据点，则在 Amazon EC2 指标上配置的警报可能会暂时进入 INSUFFICIENT_DATA 状态。这种情况很少见，但在指标报告中断时可能会发生，即使在 Amazon EC2 实例运行正常的情况下也是如此。对于配置为采取停止、终止、重启或恢复操作的 Amazon EC2 指标的警报，我们建议您将这些警报配置为将缺失的数据视为 missing，并使这些警报仅在处于 ALARM 状态时被触发。

为您的告警选择最佳选项可防止不必要和误导性的告警条件更改，还可以更准确地指示您的系统的运行状况。

> **Important**
>
> 评估 AWS/DynamoDB 命名空间中的指标的告警，始终会忽略缺失的数据，即便您为告警应该如何处理缺失的数据选择了不同的选项也是如此。当 AWS/DynamoDB 指标包含缺失数据时，评估该指标的告警仍将保持其当前状态。

在数据缺失时如何评估告警状态

每当告警评估是否更改状态时，CloudWatch 都会尝试检索高于 Evaluation Periods（评估期）指定的数量的数据点数。它尝试检索的数据点的确切数量取决于告警期限长度，以及它是基于标准分辨率指标，还是基于高分辨率指标。它尝试检索的数据点的时间范围是评估范围。

一旦 CloudWatch 检索这些数据点，便会发生以下情况：

- 如果评估范围内的数据点没有缺失，CloudWatch 将根据最近收集的数据点来评估告警。评估的数据点数等于告警的 Evaluation Periods（评估期）。在评估范围内，不需要额外的数据点，因此它们将被忽略。
- 如果评估范围内的一些数据点缺失，但是从评估范围内成功检索的现有数据点的总数等于或超过告警的 Evaluation Periods（评估期），则 CloudWatch 将根据已成功检索的最近现有数据点来评估告警状态，包括从更远的评估范围内获得的必要的额外数据点。在此情况下，您针对如何处理缺失数据而设置的值便没有必要，并且将被忽略。
- 如果评估范围内的一些数据点缺失，并且检索的实际数据点数量少于 Evaluation Periods（评估期）的告警数量，则 CloudWatch 将在缺失数据点中填写您针对如何处理缺失数据而指定的结果，然后评估该告警。但是，评估范围内的所有实时数据点都包含在评估中。CloudWatch 尽可能少地使用缺失数据点。

> **Note**
>
> 该行为的一种特殊情况是，在指标流停止后的一段时间内，CloudWatch 告警可能会反复重新评估最后一组数据点。如果告警在指标流即将停止之前更改了状态，这种重新评估可能会导致告警更改状态并重新执行操作。要缓解此行为，请使用较短时间段。

下表阐明了告警评估行为的示例。在第一个表中，Datapoints to Alarm（触发告警的数据点数）和 Evaluation Periods（评估期）均为 3。CloudWatch 在评估告警时会检索最近的 5 个数据点，以防最近的 3 个数据点中的某些数据点丢失。5 为告警的评估范围。

由于评估范围为 5，因此第 1 列显示最近的 5 个数据点。数据点显示为：右侧为最近的数据点，0 是未违例数据点，X 是违例数据点，而 - 是缺失数据点。

第 2 列显示 3 个必需数据点中缺失的个数。即使评估了最近的 5 个数据点，也只需要 3 个（Evaluation Periods（评估期）的设置）来评估告警状态。第 2 列中的数据点数是必须“填写”的数据点数（使用有关处理丢失数据的方式的设置）。

在第 3 – 6 列中，列标题是如何处理缺少数据的可能值。这些列中显示了为处理缺失数据的每种可能方法设置的告警状态。

数据点	必须填写的数据点数	缺失	忽略	违例	未违例
0 - X - X	0	OK	OK	OK	OK
- - - - 0	2	OK	OK	OK	OK
- - - - -	3	INSUFFICIENT_DATA	保留当前状态	ALARM	OK
0 X X - X	0	ALARM	ALARM	ALARM	ALARM
- - - X -	2	ALARM	保留当前状态	ALARM	OK

在上表的第二行中，即使将缺失数据视为违例，告警也会保持 OK（正常）状态，因为一个现有的数据点未违例，并且该数据点与两个被视为违例的缺失数据点一起进行评估。下次评估该告警时，如果数据仍缺失，它将变为 ALARM（告警）状态，因为未违例数据点不再处于评估范围内。

第三行（所有五个最新数据点都缺失）说明了缺失数据处理方式的各种设置对告警状态的影响。如果缺失的数据点被视为违例，告警将进入“ALARM（告警）”状态，而如果它们被视为未违例，则告警进入“OK（正常）”状态。如果忽略缺少的数据点，告警将保留缺失数据点之前的当前状态。如果缺少的数据点被认为已缺失，则告警没有足够的最新真实数据来进行评估，并变为“INSUFFICIENT_DATA（数据不足）”状态。

在第四行，告警转到 ALARM（告警）状态，因为三个最近的数据点违例，而告警的 Evaluation Periods（评估期）和 Datapoints to Alarm（触发告警的数据点数）都设为 3。在这种情况下，缺失的数据点将被忽略，并且不需要缺失数据评估方式的设置，因为有 3 个实际数据点需要评估。

第 5 行表示告警评估的特殊情况，称为提前告警状态。有关更多信息，请参阅 避免提前转换到告警状态。

在下一个表中，Period（时间段）再次设置为 5 分钟，并且 Datapoints to Alarm（触发告警的数据点数）为 2，而 Evaluation Periods（评估期）为 3。这是 2（最大为 3），M（最大为 N）告警。

评估范围为 5。这是最近检索到的数据点的最大数目，可以在某些数据点丢失的情况下使用这些数据点。

数据点	缺失数据点数	缺失	忽略	违例	未违例
0 - X - X	0	ALARM	ALARM	ALARM	ALARM
0 0 X 0 X	0	ALARM	ALARM	ALARM	ALARM
0 - X - -	1	OK	OK	ALARM	OK
- - - - 0	2	OK	OK	ALARM	OK
- - - X -	2	ALARM	保留当前状态	ALARM	OK

在第 1 行和第 2 行中，告警始终处于“ALARM（告警）”状态，因为 3 个最近的数据点中有 2 个违例。在第 2 行中，不需要评估范围内的两个最早的数据点，因为 3 个最近的数据点中无一缺失，因此这两个较旧的数据点将被忽略。

在第 3 行和第 4 行中，只有当缺失的数据被视为违例时，告警才会进入“ALARM（告警）”状态，在这种情况下，两个最近丢失的数据点都被视为违例。在第 4 行中，这两个被视为违例的缺失数据点提供了两个触发“ALARM（告警）”状态所必要的违例数据点。

第 5 行表示告警评估的特殊情况，称为提前告警状态。有关更多信息，请参阅以下 章节。

避免提前转换到告警状态

CloudWatch 告警评估包括用于尝试避免误报的逻辑，当数据出现间歇性时，告警会提前进入告警状态。上一节表中的第 5 行中显示的示例说明了这种逻辑。在这些行中，以及在以下示例中，Evaluation Periods（评估期）为 3，评估范围为 5 个数据点。Datapoints to Alarm（触发告警的数据点数）为 3，M（最大为 N）示例除外，其中 Datapoints to Alarm（触发告警的数据点数）为 2。

假设告警的最新数据是 - - - - X，其中有四个缺失的数据点，然后一个违例数据点作为最新的数据点。由于下一个数据点可能未违例，因此当数据处于 - - - - X 或者 - - - X - ，且 Datapoints to Alarm（触发告警的数据点数）为 3 时，告警不会立即进入“ALARM（告警）”状态。这样，当下一个数据点未违例并导致数据为 - - - X 0 或者 - - X - 0 时，误报得以避免。

但是，如果最后几个数据点是 - - X - - ，即使缺失的数据点被视为缺失，告警也会进入“ALARM（告警）”状态。这是因为根据告警的设计，当在数据点数量 Evaluation Periods（评估期）间最早的可用的违例数据点至少与 Datapoints to Alarm（触发告警的数据点数）的值同样早，并且所有其他较新的数据点都违例或缺失时，告警都会进入“ALARM（告警）”状态。在这种情况下，即使可

用数据点的总数小于 M（Datapoints to Alarm（触发告警的数据点数）），告警也会进入“ALARM（告警）”状态。

此告警逻辑也适用于 M（最大为 N）告警。如果评估范围内最早的违例数据点至少与 Datapoints to Alarm（触发告警的数据点数）的值同样早，并且所有最新的数据点均违例或缺失，则无论 M 值（Datapoints to Alarm（触发告警的数据点数））如何，告警都会进入“ALARM（告警）”状态。

高精度告警

如果对高精度指标设置告警，您可以指定 10 秒或 30 秒时间段的高精度告警，也可以设置 60 秒的任意倍数时间段的定期告警。高精度告警的费用较高。有关高精度指标的更多信息，请参阅 发布 自定义指标。

针对数学表达式的告警

您可以针对基于一个或多个 CloudWatch 指标的数学表达式的结果设置告警。用于告警的数学表达式可以包含多达 10 个指标。每个指标都必须使用相同的时间段。

对于基于数学表达式的告警，您可以指定您希望 CloudWatch 如何对待缺失数据点。在这种情况下，如果数学表达式没有返回该数据点的值，则认为该数据点缺失。

基于数学表达式的告警无法执行 Amazon EC2 操作。

有关指标数学表达式和语法的更多信息，请参阅 使用指标数学。

基于百分位数的 CloudWatch 告警和小数据样本

当您将某个百分位数设置为某个告警的统计数据时，您可以指定在没有数量足以进行有效的统计评估的数据时所执行的操作。您仍然可以选择让告警评估统计数据，并可以更改告警状态。或者，您也可以让告警在样本大小过小时忽略指标，并等到有足够的数据来实现统计显着性时对样本进行评估。

对于介于 0.5（含）和 1.00（不含）之间的百分位数，将在评估期内数据点的数量少于 10/（1 个百分位）时使用此设置。例如，如果 p99 百分位上某个告警的样本数少于 1000 个，则会使用此设置。对于 0 和 0.5（不含）之间的百分位数，当数据点的数量少于 10/百分位数时，将使用此设置。

CloudWatch 告警的常见功能

以下功能适用于所有 CloudWatch 告警：

- 您可以创建的告警数量没有限制。要创建或更新告警，请使用 CloudWatch 控制台、PutMetricAlarm API 操作，或 AWS CLI 中的 put-metric-alarm 命令。
- 告警名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符
- 您可以使用 CloudWatch 控制台、DescribeAlarms API 操作，或 AWS CLI 中的 describle-alarms 命令来列出任何或所有当前已配置的告警和列出任意特定状态的告警。
- 您可以使用 DisableAlarmActions 和 EnableAlarmActions API 操作或 AWS CLI 中的 disable-alarm-actions 和 enable-alarm-actions 命令来禁用或启用告警。
- 您可以使用 SetAlarmState API 操作或 AWS CLI 中的 set-alarm-state 命令将告警设为任意状态，以对其进行测试。此短暂状态变更只会持续到下一个告警比较发生之时。
- 您可以在创建自定义指标之前为该自定义指标创建告警。为了使告警有效，必须在告警定义中包含自定义指标的所有维度以及指标命名空间和指标名称。要执行此操作，您可以使用 PutMetricAlarm API 操作，或 AWS CLI 中的 put-metric-alarm命令。
- 您可以使用 CloudWatch 控制台、DescribeAlarmHistory API 操作，或 AWS CLI 中的 describe-alarm-history 命令查看告警的历史记录。CloudWatch 可将告警历史记录保存 30 天。每个状态转换都标有一个唯一的时间戳。个别情况下，一个状态变更的历史记录可能显示多个通知。通过时间戳可以确认独特的状态变更。
- 您可以在 CloudWatch 控制台的导航面板中通过 Favorites and recents（收藏夹和最近记录）选项收藏最近访问过的警报，具体方法为：将鼠标悬停在您想要收藏的警报上方，然后选择它旁边的星号。
- 告警的评估期数乘以每个评估期的长度不能超过一天。

> **Note**
>
> 在特定情况下，某些 AWS 资源不会向 CloudWatch 发送指标数据。
> 例如，Amazon EBS 可能不会发送未附加到 Amazon EC2 实例的可用卷的指标数据，因为该卷没有要监控的指标活动。如果为此类指标设置了一个告警，您可能会注意到其状态变为 `INSUFFICIENT_DATA`（数据不足）。这可能表示资源处于不活动状态，并不一定表示出现了问题。您可以指定每个告警如何处理丢失数据。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。

AWS 服务的最佳实践警报建议

CloudWatch 提供开箱即用的警报建议。我们建议您为其他 AWS 服务发布的指标创建此类 CloudWatch 警报。这些建议可以帮助您确定应为哪些指标设置警报，以遵循监控最佳实践。这些建议还建议了要设置的警报阈值。遵循这些建议可以帮助您避免错过对 AWS 基础设施的重要监控。

要查找警报建议，您可以使用 CloudWatch 控制台的“指标”部分，然后选择“警报建议”筛选开关。如果您在控制台中导航到“建议警报”，然后创建建议警报，CloudWatch 可以预先填充一些警报设置。对于某些建议警报，系统也会预先填充警报阈值。您还可以使用控制台下载建议警报的基础设施即代码警报定义，然后使用此代码在 AWS CloudFormation、AWS CLI 或 Terraform 中创建警报。

您还可以在 建议的警报 中查看推荐的警报列表。

您需要为自己创建的警报付费，费率与您在 CloudWatch 中创建的任何其他警报的费率相同。使用这些建议不会产生额外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

查找并创建建议警报

按照以下步骤查找 CloudWatch 建议您为其设置警报的指标，并可以选择创建其中一种警报。第一个过程说明了如何查找包含建议警报的指标，以及如何创建其中一种警报。

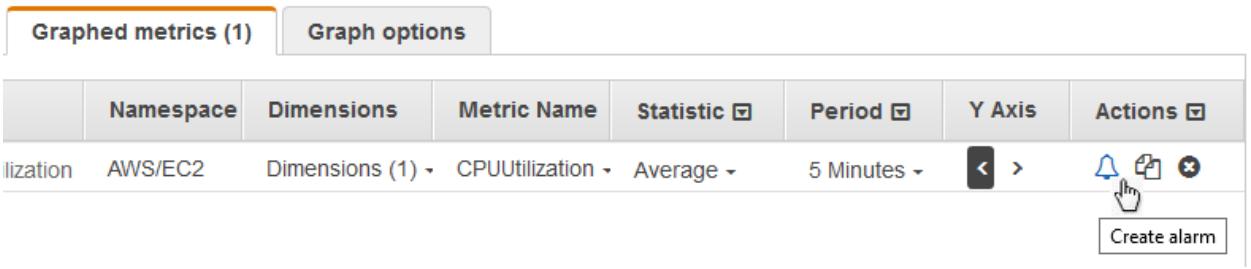
您还可以批量下载 AWS 命名空间中所有建议警报的基础设施即代码警报定义，例如 `AWS/Lambda` 或 `AWS/S3`。本主题的后续部分中包含了这些说明。

查找包含建议警报的指标，并创建单个建议警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 在指标表上方，选择警报建议。

   指标命名空间列表经过筛选，仅包含具有警报建议且您账户中的服务将发布的指标。
4. 为服务选择命名空间。

   此命名空间下的指标列表经过筛选，仅包括具有警报建议的指标。
5. 要查看指标的警报意图和建议阈值，请选择查看详细信息。
6. 要为其中一个指标创建警报，请执行以下操作之一：
   - 要使用控制台创建警报，请执行以下操作：
     a. 选中指标的复选框，然后选择图形化指标选项卡。
     b. 选择警报图标。



随即出现警报创建向导，其中根据警报建议填入了指标名称、统计数据和时间段。如果建议包含特定阈值，则该值也会预先填充。

c. 选择下一步。

d. 在通知下面，选择当警报转换为 ALARM、OK 或 INSUFFICIENT_DATA 状态时要通知的 SNS 主题。

要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

要让警报不发送通知，请选择删除。

e. 要让警报执行 Auto Scaling 或 EC2 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。

f. 在完成后，选择下一步。

g. 输入警报的名称和说明。名称只能包含 ASCII 字符。然后选择下一步。

h. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

- 要下载基础设施即代码警报定义以在 AWS CloudFormation、AWS CLI 或 Terraform 中使用，请选择下载警报代码并选择所需格式。下载的代码将包含指标名称、统计数据和阈值的建议设置。

下载针对 AWS 服务的所有建议警报的基础设施即代码警报定义

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 在指标表上方，选择警报建议。

   指标命名空间列表经过筛选，仅包含具有警报建议且您账户中的服务将发布的指标。
4. 为服务选择命名空间。

   此命名空间下的指标列表经过筛选，仅包括具有警报建议的指标。

5. 下载警报代码会显示建议为该命名空间中的指标设置多少个警报。要下载所有建议警报的基础设施即代码警报定义，请选择下载警报代码，然后选择所需代码格式。

建议的警报

以下各节列出了我们建议您为其设置最佳实践警报的指标。对于每个指标，还会显示维度、警报意图、建议阈值、阈值理由、时间段长度和数据点数量。

某些指标可能会在列表中显示两次。当建议针对该指标的不同维度组合设置不同的警报时，就会发生这种情况。

要发出警报的数据点数是必须违例才能触发警报变为“ALARM”状态的数据点数量。评估期是评估警报时考虑的时期数量。如果这些数字相同，则只有在该连续时间段的数量超过阈值时，警报才会进入“ALARM”状态。如果要发出警报的数据点数低于评估期，则属于“M（最大为 N）”警报，并且警报将进入“ALARM”状态，前提是数据点的任何评估期集合内至少有要发出警报的数据点数个数据点违例。有关更多信息，请参阅 评估告警。

主题

- Amazon API Gateway
- Amazon EC2 Auto Scaling
- Amazon CloudFront
- Amazon Cognito
- Amazon DynamoDB
- Amazon EBS
- Amazon EC2
- Amazon ElastiCache
- Amazon EC2 (AWS/ElasticGPUs)
- Amazon ECS
- 具有 Container Insights 的 Amazon ECS
- Amazon EFS
- 具有 Container Insights 的 Amazon EKS
- Amazon Kinesis Data Streams
- Lambda

- Lambda Insights
- Amazon VPC (AWS/NATGateway)
- AWS 私有链接 (AWS/PrivateLinkEndpoints)
- AWS 私有链接 (AWS/PrivateLinkServices)
- Amazon RDS
- Amazon Route 53 Public Data Plane
- Amazon S3
- S3ObjectLambda
- Amazon SNS
- Amazon SQS
- AWS VPN

Amazon API Gateway

4XXError

维度：ApiName、Stage

警报描述：此警报会检测高客户端错误率。这可能表明授权或客户端请求参数存在问题。这也可能意味着资源已被删除，或者客户端正在请求一个不存在的资源。请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致 4XX 错误的错误。此外，可以考虑启用详细的 CloudWatch 指标，以便按资源和方法查看此指标，并缩小错误来源的范围。超出配置的节流限制也可能导致错误。如果响应和日志报告的 429 错误率很高且出人意料，请按照本指南排查此问题。

意图：此警报可以检测 API Gateway 请求的高客户端错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到 4XX 错误。但是，您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 4XX 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

5XXError

维度：ApiName、Stage

警报描述：此警报有助于检测高客户端错误率。这可能表明 API 后端、网络或 API 网关与后端 API 之间的集成存在问题。本文档可以帮助您排查 5xx 错误的原因。

意图：此警报可以检测 API Gateway 请求的高服务器端错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到 5XX 错误。但是，您可以调整阈值以适应请求流量和可接受的错误率。您也可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 5XX 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

计数

维度：ApiName、Stage

警报描述：此警报有助于检测 REST API 阶段的低流量。这可能表示应用程序在调用 API 时存在问题，例如使用了错误的端点。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测 REST API 阶段的意外低流量。如果您的 API 在正常条件下收到数量可预测且一致的请求，我们建议您创建此告警。如果您已启用详细的 CloudWatch 指标，并且可以预测每个方法和资源的正常流量，我们建议您创建备用警报，以便对每种资源和方法的流量下降进行更精细的监控。对于预计流量不稳定的 API，不建议使用此告警。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准请求计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

计数

维度：ApiName、Stage、Resource、Method

警报描述：此警报有助于检测阶段中 REST API 资源和方法的低流量。这可能表示应用程序在调用 API 时存在问题，例如使用了错误的端点。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测阶段中 REST API 资源和方法的意外低流量。如果您的 API 在正常条件下收到数量可预测且一致的请求，我们建议您创建此告警。对于预计流量不稳定的 API，不建议使用此告警。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准请求计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

计数

尺寸：ApiID、Stage

警报描述：此警报有助于检测 HTTP API 阶段的低流量。这可能表示应用程序在调用 API 时存在问题，例如使用了错误的端点。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测 HTTP API 阶段的意外低流量。如果您的 API 在正常条件下收到数量可预测且一致的请求，我们建议您创建此告警。如果您已启用详细的 CloudWatch 指标，并且可以预测每个路由的正常流量，我们建议您为此创建备用警报，以便对每个路由的流量下降进行更精细的监控。对于预计流量不稳定的 API，不建议使用此告警。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准请求计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

计数

维度：ApiId、Stage、Resource、Method

警报描述：此警报有助于检测阶段中 HTTP API 路由的低流量。这可能表示应用程序在调用 API 时存在问题，例如使用了错误的端点。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测阶段中 HTTP API 路由的意外低流量。如果您的 API 在正常条件下收到数量可预测且一致的请求，我们建议您创建此告警。对于预计流量不稳定的 API，不建议使用此告警。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准请求计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

IntegrationLatency

尺寸：ApiID、Stage

警报描述：此警报有助于检测阶段中 API 请求是否存在高集成延迟。您可以将 `IntegrationLatency` 指标值与后端的相应延迟指标（例如 Lambda 集成的 `Duration` 指标）相关联。这有助于您确定 API 后端是否由于性能问题而花费更多时间来处理来自客户端的请求，或者初始化或冷启动是否存在其他开销。此外，请考虑为您的 API 启用 CloudWatch Logs，并检查日志中是否存在任何可能导致高延迟问题的错误。此外，可以考虑启用详细的 CloudWatch 指标来查看每个路由的此指标，从而帮助您缩小集成延迟来源的范围。

意图：此警报可以检测阶段中的 API Gateway 请求何时具有高集成延迟。我们建议针对 WebSocket API 设置此警报，并且认为它对于 HTTP API 来说是可选的，因为后者已经具有针对延迟指标的单独警报建议。如果您已启用详细的 CloudWatch 指标，并且每个路由的集成延迟性能要求不同，我们建议您创建备用警报，以便对每个路由的集成延迟进行更精细的监控。

统计数据：p90

建议阈值：2000.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以接受较高的延迟，则可以设置更高的阈值以降低警报的敏感度。但是，如果预计 API 能够提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后用其相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

IntegrationLatency

维度：ApiId、Stage、Route

警报描述：此警报有助于检测阶段中路由的 WebSocket API 请求是否存在高集成延迟。您可以将 `IntegrationLatency` 指标值与后端的相应延迟指标（例如 Lambda 集成的 `Duration` 指标）相关联。这有助于您确定 API 后端是否由于性能问题而花费更多时间来处理来自客户端的请求，或者初始化或冷启动是否存在其他开销。此外，请考虑为您的 API 启用 CloudWatch Logs，并检查日志中是否存在任何可能导致高延迟问题的错误。

意图：此警报可以检测阶段中路由的 API Gateway 请求何时具有高集成延迟。

统计数据：p90

建议阈值：2000.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以接受较高的延迟，则可以设置更高的阈值以降低告警的敏感度。但是，如果预计 API 能够提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后用其相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

延迟

维度：ApiName、Stage

警报描述：此警报会检测阶段中的高延迟。确定 `IntegrationLatency` 指标值以检查 API 后端延迟。如果这两个指标基本一致，则 API 后端是导致延迟更高的来源，您应该调查其中是否存在问题。另外，请考虑启用 CloudWatch Logs 并检查是否存在可能导致高延迟的错误。此外，可以考虑启用详细的 CloudWatch 指标，以便按资源和方法查看此指标，并缩小延迟来源的范围。

如果适用，请参阅如何排查与 Lambda 集成的 API Gateway 请求中的高延迟问题？或如何在 API Gateway 中解决边缘优化的 API 端点中的延迟问题？指南。

意图：此警报可以检测阶段中的 API Gateway 请求何时具有高延迟。如果您已启用详细的 CloudWatch 指标，并且每个方法和资源的延迟性能要求不同，我们建议您创建备用警报，以便对每个资源和方法的延迟进行更精细的监控。

统计数据：p90

建议阈值：2500.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以接受较高的延迟，则可以设置更高的阈值以降低告警的敏感度。但是，如果预计 API 能够提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

延迟

维度：ApiName、Stage、Resource、Method

警报描述：此警报会检测阶段中资源和方法的高延迟。确定 `IntegrationLatency` 指标值以检查 API 后端延迟。如果这两个指标基本一致，则 API 后端是导致延迟更高的来源，您应该调查其中是否存在性能问题。另外，请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致高延迟的错误。如果适用，您也可以参阅如何排查与 Lambda 集成的 API Gateway 请求中的高延迟问题？或如何在 API Gateway 中解决边缘优化的 API 端点中的延迟问题？指南。

意图：此警报可以检测阶段中资源和方法的 API Gateway 请求何时具有高延迟。

统计数据：p90

建议阈值：2500.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以

接受较高的延迟，则可以设置更高的阈值以降低告警的敏感度。但是，如果预计 API 能够提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

延迟

尺寸：ApiID、Stage

警报描述：此警报会检测阶段中的高延迟。确定 `IntegrationLatency` 指标值以检查 API 后端延迟。如果这两个指标基本一致，则 API 后端是导致延迟更高的来源，您应该调查其中是否存在性能问题。另外，请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致高延迟的错误。此外，可以考虑启用详细的 CloudWatch 指标，以便按路由查看此指标，并缩小延迟来源的范围。如果适用，您也可以参阅如何排查与 Lambda 集成的 API Gateway 请求中的高延迟问题？指南。

意图：此警报可以检测阶段中的 API Gateway 请求何时具有高延迟。如果您已启用详细的 CloudWatch 指标，并且每个路由的延迟性能要求不同，我们建议您创建备用警报，以便对每个路由的延迟进行更精细的监控。

统计数据：p90

建议阈值：2500.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以接受较高的延迟，则可以设置更高的阈值以降低其敏感度。但是，如果 API 预期会提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

延迟

维度：ApiId、Stage、Resource、Method

警报描述：此警报会检测阶段中路由的高延迟。确定 `IntegrationLatency` 指标值以检查 API 后端延迟。如果这两个指标基本一致，则 API 后端是导致延迟更高的来源，您应该调查其中是否存在性能问题。另外，请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致高延迟的错误。如果适用，您也可以参阅如何排查与 Lambda 集成的 API Gateway 请求中的高延迟问题？指南。

意图：此警报用于检测阶段中路由的 API Gateway 请求何时具有高延迟。

统计数据：p90

建议阈值：2500.0

阈值理由：建议阈值并不适用于所有 API 工作负载。但是，您可以将其用作阈值的起点。然后，您可以根据工作负载以及 API 的可接受延迟、性能和 SLA 要求选择不同的阈值。如果 API 通常可以接受较高的延迟，则可以设置更高的阈值以降低告警的敏感度。但是，如果预计 API 能够提供近乎实时的响应，请设置较低的阈值。您还可以分析历史数据以确定应用程序工作负载的预期基准延迟，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

4xx

尺寸：ApiID、Stage

警报描述：此警报会检测高客户端错误率。这可能表明授权或客户端请求参数存在问题。这也可能意味着路由已被删除，或者客户端正在请求一个 API 中不存在的资源。请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致 4xx 错误的错误。此外，可以考虑启用详细的 CloudWatch 指标，以便按路由查看此指标，帮助您缩小错误来源的范围。超出配置的节流限制也可能导致错误。如果响应和日志报告的 429 错误率很高且出人意料，请按照本指南排查此问题。

意图：此警报可以检测 API Gateway 请求的高客户端错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到 4xx 错误。但是，您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 4xx 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

5xx

尺寸：ApiID、Stage

警报描述：此警报有助于检测高客户端错误率。这可能表明 API 后端、网络或 API 网关与后端 API 之间的集成存在问题。本文档可以帮助您排查 5xx 错误的原因。

意图：此警报可以检测 API Gateway 请求的高服务器端错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到 5xx 错误。但是，您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 5xx 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

MessageCount

尺寸：ApilD、Stage

警报描述：此警报有助于检测 WebSocket API 阶段的低流量。这可能表示客户端调用 API 时存在问题（例如使用不正确的端点），或者后端向客户端发送消息时存在问题。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测 WebSocket API 阶段的意外低流量。如果您的 API 在正常条件下收到并发送数量可预测且一致的消息，我们建议您创建此警报。如果您已启用详细的 CloudWatch 指标，并且可以预测每个路由的正常流量，最好为此创建备用警报，以便对每个路由的流量下降进行更精细的监控。对于预计流量不稳定的 API，不建议使用此警报。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准消息计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

MessageCount

维度：Apild、Stage、Route

警报描述：此警报有助于检测阶段中 WebSocket API 路由的低流量。这可能表示客户端调用 API 时存在问题（例如使用不正确的端点），或者后端向客户端发送消息时存在问题。这也可能表明 API 的配置或权限存在问题，导致客户端无法访问它。

意图：此警报可检测阶段中 WebSocket API 路由的意外低流量。如果您的 API 在正常条件下收到并发送数量可预测且一致的消息，我们建议您创建此警报。对于预计流量不稳定的 API，不建议使用此警报。

统计数据：SampleCount

建议阈值：取决于您的情况

阈值理由：根据历史数据分析设置阈值，以确定您的 API 的预期基准消息计数是多少。将阈值设置为非常高的值，可能会导致警报在正常流量和预期较低流量的时间段过于敏感。相反，将其设置为非常低的值可能会导致警报错过流量异常小的下降。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

ClientError

尺寸：ApiID、Stage

警报描述：此警报会检测高客户端错误率。这可能表示授权或消息参数存在问题。这也可能意味着路由已被删除，或者客户端正在请求一个 API 中不存在的资源。请考虑启用 CloudWatch Logs 并检查是否存在任何可能导致 4xx 错误的错误。此外，可以考虑启用详细的 CloudWatch 指标，以便按路由查看此指标，帮助您缩小错误来源的范围。超出配置的节流限制也可能导致错误。如果响应和日志报告的 429 错误率很高且出人意料，请按照本指南排查此问题。

意图：此警报可以检测 WebSocket API Gateway 消息的高客户端错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到 4xx 错误。您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 4xx 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ExecutionError

尺寸：ApiID、Stage

警报描述：此警报有助于检测高执行错误率。这可能是由于您的集成出现 5xx 错误、权限问题或其他阻碍成功调用集成的因素（例如集成受限制或遭删除）所致。请考虑为您的 API 启用 CloudWatch Logs，并检查日志以了解错误的类型和原因。此外，可以考虑启用详细的 CloudWatch 指标，以便按路由查看此指标，帮助您缩小错误来源的范围。本文档可以帮助您排查任何连接错误的原因。

意图：此警报可以检测 WebSocket API Gateway 消息的高执行错误率。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值会检测何时总请求数中超过 5% 的请求会收到执行错误。您可以调整阈值以适应请求的流量以及可接受的错误率。您可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的执行错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

Amazon EC2 Auto Scaling

GroupInServiceCapacity

维度：AutoScalingGroupName

警报描述：此警报有助于检测组中的容量何时低于工作负载所需的容量。要排查问题，请检查您的扩展活动是否存在启动失败，并确认所需的容量配置是否正确。

意图：此警报可以检测由于启动失败或暂停而导致的自动扩缩组中的低可用性。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：阈值应为运行工作负载所需的最低容量。在大多数情况下，您可以将其设置为与 GroupDesiredCapicity 指标相匹配。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

Amazon CloudFront

5xxErrorRate

维度：DistributionId、Region=Global

警报描述：此警报会监控来自您的原始服务器的 5xx 错误响应百分比，帮助您检测 CloudFront 服务是否存在问题。有关帮助您了解服务器问题的信息，请参阅对来自源的错误响应进行故障排除。此外，开启其他指标可获取详细的错误指标。

意图：此警报用于检测处理来自原始服务器的请求时出现的问题，或者 CloudFront 与原始服务器之间的通信问题。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于对 5xx 响应的容忍度。您可以分析历史数据和趋势，然后相应地设置阈值。由于 5xx 错误可能由暂时性问题引起，我们建议您将阈值设置为大于 0 的值，这样警报就不会过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

OriginLatency

维度：DistributionId、Region=Global

警报描述：此警报有助于监控原始服务器的响应时间是否过长。如果服务器响应时间过长，则可能会导致超时。如果您持续遇到高 OriginLatency 值的问题，请参阅查找并修复原始服务器上来自应用程序的延迟响应。

意图：此警报用于检测原始服务器响应时间过长的问题。

统计数据：p90

建议阈值：取决于您的情况

阈值理由：您应计算原始响应超时大约 80% 的值，并将结果用作阈值。如果此指标持续接近源响应超时值，则您可能会开始遇到 504 错误。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

FunctionValidationErrors

维度：DistributionId、FunctionName、Region=Global

警报描述：此警报可帮助您监控 CloudFront Functions 中的验证错误，以便您可以采取措施解决它们。分析 CloudWatch 函数日志并查看函数代码，找出问题的根本原因并予以解决。要了解 CloudFront Functions 的常见配置错误，请参阅边缘函数的限制。

意图：此警报用于检测 CloudFront Functions 中的验证错误。

统计数据：Sum

建议阈值：0.0

阈值理由：值大于 0 表示验证错误。我们建议将阈值设置为 0，因为验证错误意味着 CloudFront Functions 交回给 CloudFront 时出现问题。例如，CloudFront 需要 HTTP 主机标头才能处理请求。没有什么可以阻止用户删除其 CloudFront Functions 代码中的主机标头。但是，当 CloudFront 返回响应并且主机标头丢失时，CloudFront 会引发验证错误。

时间段：60

要发出警报的数据点数：2

评估期：2

比较运算符：GREATER_THAN_THRESHOLD

FunctionExecutionErrors

维度：DistributionId、FunctionName、Region=Global

警报描述：此警报可帮助您监控 CloudFront Functions 中的执行错误，以便您可以采取措施解决它们。分析 CloudWatch 函数日志并查看函数代码，找出问题的根本原因并予以解决。

意图：此警报用于检测 CloudFront Functions 中的执行错误。

统计数据：Sum

建议阈值：0.0

阈值理由：我们建议将阈值设置为 0，因为执行错误表示运行时系统代码有问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

FunctionThrottles

维度：DistributionId、FunctionName、Region=Global

警报描述：此警报可帮助您监控您的 CloudFront 函数是否受到限制。如果您的函数受到限制，则意味着其执行时间太长。为避免函数限制，请考虑优化函数代码。

意图：此警报可以检测您的 CloudFront 函数何时会受到限制，以便您可以作出反应并解决问题，从而提供流畅的客户体验。

统计数据：Sum

建议阈值：0.0

阈值理由：我们建议将阈值设置为 0，以便更快地解决函数限制。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon Cognito

SignUpThrottles

维度：UserPool、UserPoolClient

警报描述：此警报会监控受限请求的数量。如果用户持续受到限制，则应通过请求增加服务限额来提高限制。要了解如何请求增加限额，请参阅 Amazon Cognito 中的限额。要主动执行操作，请考虑跟踪使用限额。

意图：此警报有助于监控受限注册请求的发生情况。这可以帮助您了解何时该执行操作来缓解注册体验的恶化。持续的限制请求会带来较差的用户注册体验。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：预置良好的用户群体不应遇到跨越多个数据点的任何限制。因此，预期工作负载的典型阈值应为 0。对于具有频繁突发的不规则工作负载，您可以分析历史数据以确定应用程序工作负载的可接受限制，然后可以相应地调整阈值。应重试受限请求，最大限度地减少对应用程序的影响。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

SignInThrottles

维度：UserPool、UserPoolClient

警报描述：此警报会监控受限用户身份验证请求的计数。如果用户持续受到限制，您可能需要通过请求增加服务限额来提高限制。要了解如何请求增加限额，请参阅 Amazon Cognito 中的限额。要主动执行操作，请考虑跟踪使用限额。

意图：此警报有助于监控受限登录请求的发生情况。这可以帮助您了解何时该执行操作来缓解登录体验的恶化。持续的限制请求会带来糟糕的用户身份验证体验。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：预置良好的用户群体不应遇到跨越多个数据点的任何限制。因此，预期工作负载的典型阈值应为 0。对于具有频繁突发的不规则工作负载，您可以分析历史数据以确定应用程序工作负载的可接受限制，然后可以相应地调整阈值。应重试受限请求，最大限度地减少对应用程序的影响。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

TokenRefreshTrott

维度：UserPool、UserPoolClient

警报描述：您可以设置阈值以适应请求的流量，并为令牌刷新请求匹配可接受限制。限制用于保护您的系统不会收到过多请求。但是，请务必监控您的正常流量是否也预置不足。您可以分析历史数据以确定应用程序工作负载的可接受限制，然后可以将警报阈值调整为高于可接受限制级别。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，非常低的阈值可能会导致警报过于敏感。

意图：此警报有助于监控受限令牌刷新请求的发生情况。这可以帮助您了解何时该执行操作来缓解任何潜在问题，从而确保流畅的用户体验以及身份验证系统的正常运行和可靠性。持续的限制请求会带来糟糕的用户身份验证体验。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：也可以设置/调整阈值，以适应请求的流量以及令牌刷新请求的可接受限制。限制是为了保护您的系统不会收到过多请求，不过，请务必监控您的正常流量是否也预置不足，并查看是否此情况是否造成了影响。还可以分析历史数据以了解应用程序工作负载的可接受限制，然后可以将阈值调整为高于一般可接受限制级别。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，非常低的阈值可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

FederationThrottles

维度：UserPool、UserPoolClient、IdentityProvider

警报描述：此警报会监控受限身份联合验证请求的计数。如果您持续受到限制，则表示您需要通过请求增加服务限额来提高限制。要了解如何请求增加限额，请参阅 Amazon Cognito 中的限额。

意图：此警报有助于监控受限身份联合验证请求的发生情况。这可以帮助您主动应对性能瓶颈或配置错误，并确保您的用户获得流畅的身份验证体验。持续的限制请求会带来糟糕的用户身份验证体验。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：您可以设置阈值以适应请求的流量，并为身份联合验证请求匹配可接受限制。限制用于保护您的系统不会收到过多请求。但是，请务必监控您的正常流量是否也预置不足。您可以分析历史数据以确定应用程序工作负载的可接受限制，然后将阈值调整为高于可接受限制级别的值。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，非常低的阈值可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon DynamoDB

AccountProvisionedReadCapacityUtilization

维度：None

警报描述：此警报会检测账户的读取容量是否将达到其预置限制。如果发生这种情况，您可以提高读取容量利用率的账户限额。您可以使用服务限额查看读取容量单位的当前限额，并请求增加限额。

意图：此警报可以检测账户的读取容量利用率是否接近其预置读取容量利用率。如果利用率达到最大限制，DynamoDB 会开始限制读取请求。

统计数据：Maximum

建议阈值：80.0

阈值理由：将阈值设置为 80%，这样就可以在阈值达到最大容量之前执行操作（例如提高账户限制），以避免限制。

时间段：300

要发出警报的数据点数：2

评估期：2

比较运算符：GREATER_THAN_THRESHOLD

AccountProvisionedWriteCapacityUtilization

维度：None

警报描述：此警报会检测账户的写入容量是否将达到其预置限制。如果发生这种情况，您可以提高写入容量利用率的账户限额。您可以使用服务限额查看写入容量单位的当前限额，并请求增加限额。

意图：此警报可以检测账户的写入容量利用率是否接近其预置写入容量利用率。如果利用率达到最大限制，DynamoDB 会开始限制写入请求。

统计数据：Maximum

建议阈值：80.0

阈值理由：将阈值设置为 80%，这样就可以在阈值达到最大容量之前执行操作（例如提高账户限制），以避免限制。

时间段：300

要发出警报的数据点数：2

评估期：2

比较运算符：GREATER_THAN_THRESHOLD

AgeOfOldestUnreplicatedRecord

维度：TableName、DelegatedOperation

警报描述：此警报会检测复制到 Kinesis 数据流过程中的延迟。在正常运行情况下，`AgeOfOldestUnreplicatedRecord` 应该只有几毫秒。根据由客户控制的配置选项造成的失败复制尝试次数，此数量会增加。例如，可能导致复制尝试失败的客户控制配置包括：预置不足的 Kinesis 数据流容量导致过度限制，或者手动更新 Kinesis 数据流的访问策略会阻止 DynamoDB 向数据流添加数据。为将此指标保持在尽可能低的水平，您需要确保预置合适的 Kinesis 数据流容量，并确保未更改 DynamoDB 的权限。

意图：此警报可以监控失败复制尝试次数，以及由此导致的复制到 Kinesis 数据流过程中的延迟。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：根据以毫秒为单位的所需复制延迟设置阈值。此值取决于您的工作负载要求和预期性能。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

FailedToReplicateRecordCount

维度：TableName、DelegatedOperation

警报描述：此警报会检测 DynamoDB 无法复制到 Kinesis 数据流的记录数。某些大于 34KB 的项目可能会扩增，以更改大于 Kinesis Data Streams 1MB 项目大小限制的数据记录。当大于 34KB 的项目包含大量布尔值或空属性值时，就会出现此扩增现象。布尔值和空属性值在 DynamoDB 中存储为 1 个字节，但在使用标准 JSON 进行 Kinesis Data Streams 复制时，最多可扩展到 5 个字节。DynamoDB 无法将此类更改记录复制到 Kinesis 数据流中。DynamoDB 跳过这些更改数据记录，并自动继续复制后续记录。

意图：此警报可以监控由于 Kinesis 数据流的项目大小限制，DynamoDB 无法复制到 Kinesis 数据流的记录数。

统计数据：Sum

建议阈值：0.0

阈值理由：将阈值设置为 0，可检测 DynamoDB 未能复制的任何记录。

时间段：60

要发出警报的数据点数：1

评估期：1

比较运算符：GREATER_THAN_THRESHOLD

ReadThrottleEvents

维度：TableName

警报描述：此警报可检测 DynamoDB 表是否有大量读取请求受到限制。要解决此问题，请参阅解决 Amazon DynamoDB 中的限制问题。

意图：此警报可以检测对 DynamoDB 表的读取请求的持续限制。持续限制读取请求会对您的工作负载读取操作产生负面影响，并降低系统的整体效率。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据 DynamoDB 表的预期读取流量设置阈值，同时考虑可接受的限制级别。请务必监控您的预置是否不足并且不会导致持续的限制。您还可以分析历史数据以确定应用程序工作负载的可接受限制级别，然后将警报阈值调整为高于一般限制级别。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，阈值过低可能会导致警报过于敏感，从而导致不必要的状态转换。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ReadThrottleEvents

维度：TableName、GlobalSecondaryIndexName

警报描述：此警报可检测 DynamoDB 表的全局二级索引是否有大量读取请求受到限制。要解决此问题，请参阅解决 Amazon DynamoDB 中的限制问题。

意图：此警报可以检测对 DynamoDB 表的全局二级索引读取请求的持续限制。持续限制读取请求会对您的工作负载读取操作产生负面影响，并降低系统的整体效率。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据 DynamoDB 表的预期读取流量设置阈值，同时考虑可接受的限制级别。请务必监控您的预置是否不足并且不会导致持续的限制。您还可以分析历史数据以确定应用程序工作负载的可接受限制级别，然后将阈值调整为高于一般可接受限制级别。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，阈值过低可能会导致警报过于敏感，从而导致不必要的状态转换。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ReplicationLatency

维度：TableName、ReceivingRegion

警报描述：此警报会检测全局表某个区域中的副本是否滞后于源区域。如果 AWS 区域降级，并且您在该区域有一个副本表，则延迟可能会增加。在这种情况下，可以临时将应用程序的读取和写入活动重定向到不同的 AWS 区域。如果您使用的是 2017.11.29（旧版）的全局表，则应验证每个副本表的写入容量单位（WCU）是否相同。您也可以确保遵循 Best practices and requirements for managing capacity 中的建议。

意图：此警报可以检测一个区域中的副本表是否落后于从另一个区域复制更改。这可能会导致您的副本与其他副本不同。了解每个 AWS 区域的复制延迟，并在该复制延迟持续增加时发出提醒会很有帮助。表的复制仅适用于全局表。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于您的用例。超过 3 分钟的复制延迟通常都需要调查。查看复制延迟的临界程度和要求并分析历史趋势，然后相应地选择阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

SuccessfulRequestLatency

维度：TableName、Operation

警报描述：此警报会检测 DynamoDB 表操作的高延迟（由警报中 `Operation` 的维度值表示）。请参阅此问题排查文档，了解如何排查 Amazon DynamoDB 中的延迟问题。

意图：此警报可以检测 DynamoDB 表操作的高延迟。操作的较高延迟会对系统的整体效率产生负面影响。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：DynamoDB 会为单例操作（例如 getItem、PutiTem 等）提供平均个位数毫秒延迟。但是，您可以根据工作负载中涉及的操作类型和表的可接受延迟容差来设置阈值。您可以分析此指标的历史数据以确定表操作的一般延迟，然后将阈值设置为代表操作的临界延迟的数字。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

SystemErrors

维度：TableName

警报描述：此警报会检测 DynamoDB 表请求的持续大量系统错误。如果您持续收到 5xx 错误，请打开 AWS 服务运行状况控制面板，以检查服务是否存在操作问题。如果 DynamoDB 持续存在内部

服务问题，您可以使用此警报来获取通知，它可以帮助您弄清楚您的客户端应用程序面临的问题。有关更多信息，请参阅 DynamoDB 错误处理。

意图：此警报可以检测 DynamoDB 表请求的持续系统错误。系统错误表示 DynamoDB 中的内部服务错误，有助于您弄清楚客户端遇到的问题。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据预期流量设置阈值，同时考虑系统错误的可接受级别。您还可以分析历史数据以确定应用程序工作负载的可接受错误计数，然后相应地调整阈值。系统错误应由应用程序/服务重试，因为它们是暂时性的。因此，阈值过低可能会导致警报过于敏感，从而导致不必要的状态转换。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

ThrottledPutRecordCount

维度：TableName、DelegatedOperation

警报描述：此警报会检测在将更改数据捕获复制到 Kinesis 的过程中，您的 Kinesis 数据流限制的记录。出现此限制的原因是 Kinesis 数据流容量不足。如果遇到过多和定期的限制，则可能需要按照观察到的表写入吞吐量成比例增加 Kinesis 流分片数量。要了解有关如何确定 Kinesis 数据流的大小的详细信息，请参阅确定 Kinesis Data Streams 的初始大小。

意图：此警报可以监控由于 Kinesis 数据流容量不足而受到 Kinesis 数据流限制的记录数量。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：异常使用高峰期间可能会遇到一些限制，但受限记录应保持在尽可能低的水平，以避免较高的复制延迟（DynamoDB 重试将受限记录发送到 Kinesis 数据流）。将阈值设置为一个数字，这样可以帮助您发现常规过度限制。您还可以分析此指标的历史数据，以确定应用程序工作负载的可接受限制速率。根据您的用例将阈值调整为应用程序可以容忍的值。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

UserErrors

维度：None

警报描述：此警报会检测 DynamoDB 表请求的持续大量用户错误。您可以在问题时间范围内查看客户端应用程序日志，了解请求无效的原因。您可以检查 HTTP 状态码 400，查看您收到的错误类型并执行相应操作。您可能必须修复应用程序逻辑才能创建有效的请求。

意图：此警报可以检测 DynamoDB 表请求的持续用户错误。所请求操作的用户错误表示客户端在生成无效请求并且正在失败。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：将阈值设置为 0 可检测任何客户端错误。或者，如果要避免因错误数量非常少而触发警报，可以将其设置为更高的值。基于您的用例和请求的流量来决定。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

WriteThrottleEvents

维度：TableName

警报描述：此警报可检测 DynamoDB 表是否有大量写入请求受到限制。要解决此问题，请参阅解决 Amazon DynamoDB 中的限制问题。

意图：此警报可以检测 DynamoDB 表的写入请求的持续限制。持续限制写入请求会对您的工作负载写入操作产生负面影响，并降低系统的整体效率。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据 DynamoDB 表的预期写入流量设置阈值，同时考虑可接受的限制级别。请务必监控您的预置是否不足并且不会导致持续的限制。您还可以分析历史数据以确定应用程序工作负载的可接受限制级别，然后将阈值调整为高于一般可接受限制级别的值。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，阈值过低可能会导致警报过于敏感，从而导致不必要的状态转换。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

WriteThrottleEvents

维度：TableName、GlobalSecondaryIndexName

警报描述：此警报可检测 DynamoDB 表的全局二级索引是否有大量写入请求受到限制。要解决此问题，请参阅解决 Amazon DynamoDB 中的限制问题。

意图：此警报可以检测对 DynamoDB 表全局二级索引的写入请求的持续限制。持续限制写入请求会对您的工作负载写入操作产生负面影响，并降低系统的整体效率。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据 DynamoDB 表的预期写入流量设置阈值，同时考虑可接受的限制级别。请务必监控您的预置是否不足并且不会导致持续的限制。您还可以分析历史数据以确定应用程序工作负载的可接受限制级别，然后将阈值调整为高于一般可接受限制级别的值。受限请求应由应用程序/服务重试，因为它们是暂时性的。因此，值过低可能会导致警报过于敏感，从而导致不必要的状态转换。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon EBS

VolumeStalledIOCheck

维度：VolumeId, InstanceId

警报描述：此警报有助于监控 Amazon EBS 卷的 IO 性能。此检查可检测 Amazon EBS 基础设施的潜在问题，例如 Amazon EBS 卷所基于的存储子系统硬件或软件问题、影响从 Amazon EC2 实例访问 Amazon EBS 卷的物理主机硬件问题，还可以检测实例与 Amazon EBS 卷之间的连接问题。如果“停滞 IO 检查”失败，您可以等待 AWS 解决问题，也可以自行采取措施，例如替换受影响的卷或停止并重启挂载了该卷的实例。在大多数情况下，当该指标失败时，Amazon EBS 将在几分钟内自动诊断并恢复您的卷。

意图：此警报可以检测 Amazon EBS 卷的状态，以确定这些卷何时受损且无法完成 I/O 操作。

统计数据：Maximum

建议阈值：1.0

阈值理由：当状态检查失败时，此指标的值为 1。设置阈值后，每当状态检查失败时，警报都将处于“ALARM”状态。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EC2

CPU 利用率

维度：InstanceId

警报描述：此警报有助于监控 EC2 实例的 CPU 利用率。根据应用程序的不同，持续的高利用率级别可能是正常的。但是，如果性能下降，并且应用程序不受磁盘 I/O、内存或网络资源的限制，则达到极限的 CPU 可能表示存在资源瓶颈或应用程序性能问题。高 CPU 利用率可能表示需要升级到

CPU 密集型实例。如果启用了详细监控，则可以将时间段更改为 60 秒而不是 300 秒。有关更多信息，请参阅对实例启用或禁用详细监控。

意图：此警报用于检测高 CPU 利用率。

统计数据：平均值

建议阈值：80.0

阈值理由：通常，您可以将 CPU 利用率的阈值设置为 70-80%。但是，您可以根据可接受性能级别和工作负载特征来调整此值。对于某些系统，持续的高 CPU 利用率可能是正常现象，并不表示存在问题，而对于某些系统，此情况应引起注意。分析历史 CPU 利用率数据以确定使用情况，弄清您系统的可接受 CPU 利用率，并相应地设置阈值。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

StatusCheckFailed

维度：InstanceId

警报描述：此警报有助于监控系统状态检查和实例状态检查。如果任一类型的状态检查失败，则此警报应处于“ALARM”状态。

意图：此警报用于检测实例的根本问题，包括系统状态检查失败和实例状态检查失败。

统计数据：Maximum

建议阈值：1.0

阈值理由：当状态检查失败时，此指标的值为 1。设置阈值后，每当状态检查失败时，警报都将处于“ALARM”状态。

时间段：300

要发出警报的数据点数：2

评估期：2

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

StatusCheckFailed_AttachedEBS

维度：InstanceId

警报描述：此警报可帮助您监控附加到实例的 Amazon EBS 卷是否可以访问并能够完成 I/O 操作。此状态检查可检测计算或 Amazon EBS 基础设施存在的底层问题，如下所示：

- Amazon EBS 卷底层的存储子系统的硬件或软件问题
- 物理主机上的硬件问题，该问题会影响 Amazon EBS 卷的可访问性
- 实例和 Amazon EBS 卷之间的连接问题

当附加的 EBS 状态检查失败时，您可以等待 Amazon 解决问题，也可以自行采取措施，例如更换受影响的卷或停止并重启实例。

意图：此警报用于检测附加到实例的哪些 Amazon EBS 卷无法访问。这些会导致 I/O 操作失败。

统计数据：Maximum

建议阈值：1.0

阈值理由：当状态检查失败时，此指标的值为 1。设置阈值后，每当状态检查失败时，警报都将处于“ALARM”状态。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon ElastiCache

CPU 利用率

维度：CacheClusterId、CacheNodeId

警报描述：此警报有助于监控整个 ElastiCache 实例的 CPU 利用率，包括数据库引擎进程和实例上运行的其他进程。AWSElasticache 支持两种引擎类型：Memcached 和 Redis OSS。当您在 Memcached 节点上达到高 CPU 利用率时，应考虑纵向扩展实例类型或添加新的缓存节点。对于

Redis OSS，如果您的主要工作负载来自读取请求，则应考虑向缓存集群添加更多只读副本。如果您的主要工作负载来自写入请求，则要在集群模式下运行时，应考虑添加更多分片以将工作负载分配到更多主节点；而要非集群模式下运行 Redis OSS 时，应考虑纵向扩展实例类型。

意图：此警报用于检测 ElastiCache 主机的高 CPU 利用率。这对全面了解整个实例（包括非引擎进程）的 CPU 使用情况很有用。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：将阈值设置为反映应用程序的临界 CPU 利用率级别的百分比。对于 Memcached，引擎最多可以使用 num_threads 个核心。对于 Redis OSS，引擎主要是单线程的，但如果有额外的核心，也可以使用它们来加速 I/O。在大多数情况下，您可以将阈值设置为可用 CPU 的 90% 左右。因为 Redis OSS 是单线程的，实际阈值应计算为节点总容量的一小部分。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

CurrConnections

维度：CacheClusterId、CacheNodeId

警报描述：此警报可检测高连接计数，这可能表示负载过重或存在性能问题。`CurrConnections` 的持续增加可能会导致 65000 个可用连接耗尽。这可能表明应用程序端的连接关闭不当，并且在服务器端未断开。您应该考虑使用连接池或空闲连接超时来限制与集群建立的连接数量，或者对于 Redis OSS，可以考虑在集群上调整 tcp-keepalive，以检测和终止潜在失效对端。

意图：此警报有助于您识别可能影响 ElastiCache 集群性能和稳定性的高连接计数。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于集群的可接受连接范围。查看 ElastiCache 集群的容量和预期工作负载，分析常规使用期间的历史连接计数以建立基准，然后相应地选择阈值。请记住，每个节点最多可以支持 65000 个并发连接。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

DatabaseMemoryUsagePercentage

维度：CacheClusterId

警报描述：此警报有助于监控集群的内存利用率。当 `DatabaseMemoryUsagePercentage` 达到 100% 时，将触发 Redis OSS maxmemory 策略，并且可能会根据所选策略进行驱逐。如果缓存中没有符合驱逐策略的对象，则写入操作将失败。有些工作负载期望或依赖驱逐，如果不是这样，您需要增加集群的内存容量。您可以通过添加更多主节点来横向扩展集群，也可以使用更大的节点类型对其纵向扩展。有关详细信息，请参阅 Scaling ElastiCache for Redis OSS clusters。

意图：此警报用于检测集群的高内存利用率，以便在写入集群时避免失败。如果您的应用程序预计不会遭遇驱逐，则了解何时需要纵向扩展集群会很有用。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：根据应用程序的内存要求和 ElastiCache 集群的内存容量，您应将阈值设置为反映集群内存使用量临界水平的百分比。您可以使用历史内存使用数据，作为可接受内存使用量阈值的参考。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

EngineCPUUtilization

维度：CacheClusterId

警报描述：此警报有助于监控 ElastiCache 实例中 Redis OSS 引擎线程的 CPU 利用率。引擎 CPU 占用率高的常见原因包括：消耗高 CPU 的长时间运行的命令、大量请求、短时间内新的客户端连

接请求增加，以及缓存没有足够的内存来容纳新数据时的高驱逐率。您应该考虑通过添加更多节点或纵向扩展实例类型来实现 ElastiCache for Redis OSS 集群的扩缩。

意图：此警报用于检测 Redis OSS 引擎线程的高 CPU 利用率。它在要监控数据库引擎本身的 CPU 利用率时很有用。

统计数据：平均值

建议阈值：90.0

阈值理由：将阈值设置为反映应用程序的临界引擎 CPU 利用率级别的百分比。您可以使用应用程序和预期工作负载对集群进行基准测试，关联 EngineCPUUtilization 和性能作为参考，然后相应地设置阈值。在大多数情况下，您可以将阈值设置为可用 CPU 的 90% 左右。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ReplicationLag

维度：CacheClusterId

警报描述：此警报有助于监控您的 ElastiCache 集群的复制运行状况。高复制滞后意味着主节点或副本无法跟上复制的步伐。如果您的写入活动过高，可以考虑通过添加更多主节点来横向扩展集群，或者使用更大的节点类型纵向扩展集群。有关详细信息，请参阅 Scaling ElastiCache for Redis OSS clusters。如果您的只读副本因读取请求的数量而过载，可以考虑添加更多只读副本。

意图：此警报用于检测主节点上的数据更新与其同步到副本节点之间的延迟。它有助于确保只读副本集群节点的数据一致性。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：根据应用程序的要求和复制滞后的潜在影响来设置阈值。您应该考虑应用程序的预期写入速率和网络条件，以确定可接受的复制滞后。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

Amazon EC2 (**AWS/ElasticGPUs**)

GPUConnectivityCheckFailed

维度：InstanceId、EGPUId

警报描述：此警报有助于检测实例与 Elastic Graphics 加速器之间的连接故障。Elastic Graphics 使用实例网络将 OpenGL 命令发送到远程附加的显卡。此外，运行带有 Elastic Graphics 加速器的 OpenGL 应用程序的桌面通常使用远程访问技术来访问。确定性能问题是与 OpenGL 渲染相关还是与桌面远程访问技术相关，这一点非常重要。要了解有关该问题的更多信息，请参阅调查应用程序性能问题。

意图：此警报用于检测从实例到 Elastic Graphics 加速器的连接问题。

统计数据：Maximum

建议阈值：0.0

阈值理由：阈值为 1 表示连接已失败。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

GPUHealthCheckFailed

维度：InstanceId、EGPUId

警报描述：此警报有助于您了解 Elastic Graphics 加速器的状态何时不正常。如果加速器运行状况不佳，请参阅解决不正常状态问题中的问题排查步骤。

意图：此警报用于检测 Elastic Graphics 加速器是否运行状况不佳。

统计数据：Maximum

建议阈值：0.0

阈值理由：阈值为 1 表示状态检查失败。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

Amazon ECS

CPUReservation

维度：ClusterName

警报描述：此警报有助于您检测 ECS 集群的高 CPU 预留。高 CPU 预留可能表示集群将耗尽为任务注册的 CPU。要排查问题，您可以添加更多容量、扩展集群，也可以设置自动扩缩。

意图：该警报用于检测集群上任务预留的 CPU 单元总数是否达到为集群注册的 CPU 单位总数。这有助于您了解何时纵向扩展集群。达到集群的 CPU 单位总数可能会导致任务的 CPU 耗尽。如果您开启了 EC2 容量提供商托管式扩展，或者您已将 Fargate 关联到容量提供商，则不建议设置此警报。

统计数据：平均值

建议阈值：90.0

阈值理由：将 CPU 预留阈值设置为 90%。或者，您可以基于集群特征选择较小的值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

CPU 利用率

维度：ClusterName、ServiceName

警报描述：此警报有助于您检测 ECS 服务的高 CPU 利用率。如果没有正在进行的 ECS 部署，则达到极限的 CPU 利用率可能表示存在资源瓶颈或应用程序性能问题。要排查问题，可以提高 CPU 限制。

意图：此警报用于检测 ECS 服务的高 CPU 利用率。持续的高 CPU 利用率可能表示存在资源瓶颈或应用程序性能问题。

统计数据：平均值

建议阈值：90.0

阈值理由：CPU 利用率的服务指标可能超过 100% 的利用率。但是，我们建议您监控高 CPU 利用率的指标，避免影响其他服务。将阈值设置为大约 90-95%。我们建议您更新任务定义以反映实际使用量，以防其他服务将来出现问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

MemoryReservation

维度：ClusterName

警报描述：此警报有助于您检测 ECS 集群的高内存预留。高内存预留可能表示集群存在资源瓶颈。要排查问题，请分析服务任务的性能，查看是否可以优化任务的内存利用率。此外，您可以注册更多内存或设置自动扩缩。

意图：该警报用于检测集群上任务预留的存储单元总数是否达到为集群注册的存储单元总数。这有助于您了解何时纵向扩展集群。达到集群的存储单元总数可能会导致集群无法启动新任务。如果您开启了 EC2 容量提供商托管式扩展，或者您已将 Fargate 关联到容量提供商，则不建议设置此警报。

统计数据：平均值

建议阈值：90.0

阈值理由：将内存预留阈值设置为 90%。您可以基于集群特征将其调整为较小的值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

HTTPCode_Target_5XX_Count

维度：ClusterName、ServiceName

警报描述：此警报有助于您检测 ECS 服务的服务器端高错误计数。这可能表示存在导致服务器无法处理请求的错误。要排查问题，请检查您的应用程序日志。

意图：此警报用于检测 ECS 服务的服务器端高错误计数。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：计算约占平均流量的 5% 的值，并使用该值作为阈值的起点。您可以使用 `RequestCount` 指标确定平均流量。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。需要对经常发生的 5XX 错误发出警报。但是，将阈值设置得非常低可能会导致警报过于敏感。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

TargetResponseTime

维度：ClusterName、ServiceName

警报描述：此警报有助于您检测 ECS 服务请求的长目标响应时间。这可能表示存在导致服务无法及时处理请求的错误。要排查问题，请检查 CPUUtilization 指标以查看服务是否耗尽了 CPU，或者检查您的服务所依赖的其他下游服务的 CPU 利用率。

意图：此警报用于检测 ECS 服务请求的长目标响应时间。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于您的用例。查看服务的目标响应时间的临界程度和要求，并分析此指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

具有 Container Insights 的 Amazon ECS

EphemeralStorageUtilized

维度：ClusterName、ServiceName

警报描述：此警报有助于您检测 Fargate 集群的临时存储空间利用率较高的情况。如果临时存储空间利用率持续较高，则可以检查临时存储空间使用情况并增加临时存储空间。

意图：此警报用于检测 Fargate 集群的临时存储空间利用率较高的情况。如果临时存储空间利用率持续较高，则可能表明磁盘已满，并可能导致容器出现故障。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：将阈值设置为临时存储空间大小的 90% 左右。您可以根据您的 Fargate 集群可接受的临时存储空间利用率调整此值。对于某些系统，临时存储空间利用率持续较高可能是正常的，而对于另一些系统，则可能会导致容器出现故障。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

RunningTaskCount

维度：ClusterName、ServiceName

警报描述：此警报有助于您检测 ECS 服务的运行任务数不足。如果正在运行的任务数太少，则可能表明应用程序无法处理服务负载，并可能导致性能问题。如果没有正在运行的任务，Amazon ECS 服务可能不可用或者可能存在部署问题。

意图：此警报用于检测正在运行的任务数是否过少。持续较少的运行任务数可能表明 ECS 服务部署或性能存在问题。

统计数据：平均值

建议阈值：0.0

阈值调整：您可以根据 ECS 服务的最小运行任务数来调整阈值。如果正在运行的任务数为 0，则 Amazon ECS 服务将不可用。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_OR_EQUAL_TO_THRESHOLD

instance_filesystem_utilization

维度：InstanceId、ContainerInstanceId、ClusterName

警报描述：此警报有助于您检测 ECS 集群的文件系统利用率较高的情况。如果文件系统利用率持续较高，请检查磁盘利用率。

意图：此警报用于检测 Amazon ECS 集群的文件系统利用率较高的情况。文件系统利用率持续较高可能表示存在资源瓶颈或应用程序性能问题，并且可能会阻碍新任务的运行。

统计数据：平均值

建议阈值：90.0

阈值调整：您可以将文件系统利用率的阈值设置为 90-95%。您可以根据 Amazon ECS 集群可接受的文件系统容量级别调整此值。对于某些系统，文件系统利用率持续较高可能是正常现象，并不表示存在问题；而对于另一些系统，这可能令人担忧，并可能导致性能问题并阻碍新任务的运行。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon EFS

PercentIOLimit

维度：FileSystemId

警报描述：此警报有助于确保工作负载保持在文件系统可用的 I/O 限制范围内。如果指标持续达到其 I/O 限制，可以考虑将应用程序移至使用最大 I/O 性能作为模式的文件系统。要排查问题，请检查连接到文件系统的客户端和限制文件系统之客户端的应用程序。

意图：此警报用于检测文件系统接近通用性能模式的 I/O 限制的情况。持续的高 I/O 百分比可能表明文件系统无法在 I/O 请求方面进行足够的扩展，并且文件系统可能成为使用该文件系统之应用程序的资源瓶颈。

统计数据：平均值

建议阈值：100.0

阈值理由：当文件系统达到其 I/O 限制时，它对读取和写入请求的响应速度可能会变慢。因此，建议监控该指标，以免影响使用文件系统的应用程序。阈值可以设置为 100% 左右。但是，可根据文件系统特征将此值调整为较低的值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

BurstCreditBalance

维度：FileSystemId

警报描述：此警报有助于确保文件系统使用量有可用的突增积分余额。当没有可用的突增积分时，由于吞吐量低，应用程序对文件系统的访问将受到限制。如果指标持续降至 0，可以考虑将吞吐量模式更改为 Elastic 或预置吞吐量模式。

意图：此警报用于检测文件系统的低突增积分余额。持续的低突增积分余额可能表明吞吐量降低和 I/O 延迟增加。

统计数据：平均值

建议阈值：0.0

阈值理由：当文件系统耗尽突增积分时，即使基准吞吐率较低，EFS 仍会继续向所有文件系统提供 1MiBps 的计量吞吐量。但是，建议监控指标是否存在低突增积分余额，避免文件系统成为应用程序的资源瓶颈。阈值可以设置为 0 字节左右。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：LESS_THAN_OR_EQUAL_TO_THRESHOLD

具有 Container Insights 的 Amazon EKS

node_cpu_utilization

维度：ClusterName

警报描述：此警报有助于检测 EKS 集群的 Worker 节点中的高 CPU 利用率。如果利用率一直很高，则可能表明需要将 Worker 节点替换为具有更高 CPU 的实例，或者需要横向扩展系统。

意图：此警报有助于监控 EKS 集群中 Worker 节点的 CPU 利用率，这样系统性能就不会降低。

统计数据：Maximum

建议阈值：80.0

阈值理由：建议将阈值设置为小于或等于 80%，以便有足够的时间在系统开始受到影响之前调试问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

node_filesystem_utilization

维度：ClusterName

警报描述：此警报有助于检测 EKS 集群的 Worker 节点中的高文件系统利用率。如果利用率一直很高，则可能需要更新 Worker 节点以拥有更大的磁盘卷，或者可能需要横向扩展。

意图：此警报有助于监控 EKS 集群中 Worker 节点的文件系统利用率。如果利用率达到 100%，则可能导致应用程序故障、磁盘 I/O 瓶颈、容器组（pod）驱逐或节点完全无响应。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：如果有足够的磁盘压力（这意味着磁盘将满），则节点将被标记为运行状况不佳，容器组（pod）将被驱逐出节点。当可用文件系统低于 kubelet 上设置的驱逐阈值时，有磁盘压力的节点上的容器组（pod）将被驱逐。设置警报阈值，以便您在节点被驱逐出集群之前有足够的时间作出反应。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

node_memory_utilization

维度：ClusterName

警报描述：此警报有助于检测 EKS 集群的 Worker 节点中的高内存利用率。如果利用率一直很高，则可能表明需要扩展容器组（pod）副本的数量或优化您的应用程序。

意图：此警报有助于监控 EKS 集群中 Worker 节点的内存利用率，这样系统性能就不会降低。

统计数据：Maximum

建议阈值：80.0

阈值理由：建议将阈值设置为小于或等于 80%，以便有足够的时间在系统开始受到影响之前调试问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

pod_cpu_utilization_over_pod_limit

维度：ClusterName、Namespace、Service

警报描述：此警报有助于检测 EKS 集群的容器组（pod）中的高 CPU 利用率。如果利用率一直很高，则可能表明需要增加受影响容器组（pod）的 CPU 限制。

意图：此警报有助于监控 EKS 集群中属于 Kubernetes 服务的容器组（pod）的 CPU 利用率，以便您可以快速识别服务的容器组（pod）占用的 CPU 是否高于预期。

统计数据：Maximum

建议阈值：80.0

阈值理由：建议将阈值设置为小于或等于 80%，以便有足够的时间在系统开始受到影响之前调试问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

pod_memory_utilization_over_pod_limit

维度：ClusterName、Namespace、Service

警报描述：此警报有助于检测 EKS 集群的容器组（pod）中的高内存利用率。如果利用率一直很高，则可能表明需要增加受影响容器组（pod）的内存限制。

意图：此警报有助于监控 EKS 集群中容器组（pod）的内存利用率，这样系统性能就不会降低。

统计数据：Maximum

建议阈值：80.0

阈值理由：建议将阈值设置为小于或等于 80%，以便有足够的时间在系统开始受到影响之前调试问题。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

维度：StreamName

警报描述：此警报可以检测迭代器的最大寿命是否过高。对于实时数据处理应用程序，可以根据延迟容差配置数据留存。这通常只需几分钟就能完成。对于处理历史数据的应用程序，可以使用此指标来监控追赶速度。阻止数据丢失的快速解决方案是在排查问题时延长保留期。您还可以增加在消费端应用程序中处理记录的工作程序数量。逐渐增长迭代器寿命最常见的原因是没有足够的物理资源，或者记录处理逻辑没有随着流吞吐量的增大而相应扩展。有关更多详细信息，请参阅链接。

意图：此警报用于检测流中的数据是否会因为保存时间过长或记录处理速度太慢而过期。它可以帮助您在达到 100% 的流保留时间后避免数据丢失。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于流的保留期以及记录的处理延迟容差。查看您的需求并分析历史趋势，然后将阈值设置为表示临界处理延迟的毫秒数。如果迭代器的寿命超过了保留期的 50%（默认值为 24 小时，可配置为最高 365 天），则存在由于记录过期造成数据丢失的风险。您可以监控该指标，确保您的任何分片都不会达到此限制。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

GetRecords.Success

维度：StreamName

警报描述：每当您的消费端成功从您的流中读取数据时，此指标就会增加。
当它引发异常时，GetRecords 不会返回任何数据。最常见的例外情况是 ProvisionedThroughputExceededException，由于流的请求速率太高，或者因为给定秒钟的可用吞吐量已经得到满足。降低请求的频率或大小。有关更多信息，请参阅《Amazon Kinesis Data Streams 开发人员指南》中的 Streams Limits 和 Error Retries and Exponential Backoff in AWS。

意图：此警报可以检测消费端从流中检索记录是否失败。通过为此指标设置警报，您可以主动检测数据消耗方面的任何问题，例如错误率增加或检索成功率下降。这让您可以及时执行操作来解决潜在问题，并保持数据处理管道的顺畅运行。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：根据从流中检索记录的重要性，基于应用程序对失败记录的容忍度来设置阈值。阈值应为成功操作的相应百分比。您可以使用历史 GetRecords 指标数据作为可接受失败率的参考。设置阈值时还应考虑重试，因为可以重试失败记录。这有助于防止暂时性峰值触发不必要的提醒。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

PutRecord.Success

维度：StreamName

警报描述：此警报会检测失败的 PutRecord 操作的数量何时超过阈值。调查数据生成器日志，弄清楚失败的根本原因。最常见的原因是导致 ProvisionedThroughputExceededException 的分片的预置吞吐量不足。之所以发生这种情况，是因为流的请求速率太高，或者尝试摄取到分片中的吞吐量太高。降低请求的频率或大小。有关更多信息，请参阅 Streams Limits 和 Error Retries and Exponential Backoff in AWS。

意图：此警报可以检测向流中摄取记录是否失败。它有助于您识别向流写入数据时存在的问题。通过为此指标设置警报，您可以主动检测生成器在向流发布数据时存在的任何问题，例如错误率增加

或成功发布的记录减少。这让您可以及时执行操作来解决潜在问题，并保持数据摄取过程的可靠运行。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：根据对服务进行数据摄取和处理的重要性，基于应用程序对失败记录的容忍度来设置阈值。阈值应为成功操作的相应百分比。您可以使用历史 PutRecord 指标数据作为可接受失败率的参考。设置阈值时还应考虑重试，因为可以重试失败记录。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

PutRecords.FailedRecords

维度：StreamName

警报描述：此警报会检测失败的 `PutRecords` 的数量何时超过阈值。Kinesis Data Streams 会尝试处理每个 `PutRecords` 请求中的所有记录，但是单个记录失败并不会停止对后续记录的处理。这些失败的主要原因是超过了流或单个分片的吞吐量。常见的原因是流量激增和网络延迟，这会导致记录不均匀地到达流。您必须检测处理不成功的记录并在后续调用中重试它们。有关更多详细信息，请参阅 Handling Failures When Using PutRecords。

意图：当使用批处理操作将记录放入流时，此警报可以检测持续失败。通过为此指标设置警报，您可以主动检测失败记录的增加，从而能够及时执行操作来解决潜在问题，并确保数据摄取过程顺畅可靠。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：将阈值设置为失败记录数，以反映应用程序对失败记录的容忍度。您可以使用历史数据作为可接受失败值的参考。设置阈值时还应考虑重试，因为可以在后续 PutRecords 调用中重试失败记录。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ReadProvisionedThroughputExceeded

维度：StreamName

警报描述：此警报会跟踪导致读取吞吐能力限制的记录数。如果您发现自己持续受到限制，则应考虑向流中添加更多分片以增加预置读取吞吐量。如果有多个消费端应用程序在流中运行，并且它们共享 GetRecords 限制，我们建议您通过增强型扇出功能注册新的消费端应用程序。如果添加更多分片不会降低限制的数量，则可能有一个“热”分片被读取的次数超过了其他分片。启用增强监控，确定“热”分片，然后将其拆分。

意图：此警报可以检测消费端在超过预置的读取吞吐量（由您拥有的分片数量决定）时是否受到限制。在这种情况下，您将无法从流中读取内容，并且流可以开始备份。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：通常可以重试受限请求，因此将阈值设置为 0 会使警报过于敏感。但是，持续的限制可能会影响从流中进行读取，因此应该会触发警报。根据应用程序的限制请求和重试配置，将阈值设置为百分比。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

SubscribeToShardEvent.MillisBehindLatest

维度：StreamName、ConsumerName

警报描述：此警报会检测应用程序中的记录处理延迟何时超过阈值。诸如对下游应用程序的 API 操作失败之类的暂时性问题可能会导致指标突然增加。您应该调查它们是否持续发生。一个常见的原因是，由于物理资源不足，或者记录处理逻辑没有随着流吞吐量的增加而扩展，导致消费端处理记录的速度不够快。在关键路径中阻止调用通常是导致记录处理速度减慢的原因。您可以通过增加分片的数量来提高并行度。您还应确认底层处理节点在需求高峰期间有足够的物理资源。

意图：此警报可以检测流分片事件订阅的延迟。这表明存在处理延迟，可以帮助识别消费端应用程序性能或流的整体运行状况的潜在问题。当处理延迟变得严重时，您应该调查并解决任何瓶颈或消费端应用程序效率低下的问题，以确保实时数据处理并最大限度地减少数据积压。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于您的应用程序可以容忍的延迟。查看应用程序的要求并分析历史趋势，然后相应地选择阈值。当 SubscribeToShard 调用成功后，您的消费端将开始通过持续连接接收 SubscribeToShardEvent 事件，最长可持续 5 分钟，之后如果您想继续接收记录，需要再次调用 SubscribetoShard 来续订订阅。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

WriteProvisionedThroughputExceeded

维度：StreamName

警报描述：此警报会检测导致写入吞吐能力限制的记录数何时达到阈值。当您的生成器超过预置写入吞吐量（由您拥有的分片数量决定）时，它们就会受到限制，您将无法将记录放入流中。为了解决持续限制的问题，您应该考虑向流中添加分片。这会提高您的预置写入吞吐量并防止未来发生限制。在摄取记录时，还应考虑分区键的选择。首选随机分区键，因为它会尽可能将记录均匀地分布在流的分片上。

意图：此警报可以检测您的生成器是否因为流或分片的限制而在写入记录时被拒。如果您的数据流处于预置模式，则设置此警报有助于您在数据流达到限制时主动执行操作，让您可以优化预置容量或执行相应扩展操作，以免数据丢失并保持数据处理的顺畅运行。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：通常可以重试受限请求，因此将阈值设置为 0 会使警报过于敏感。但是，持续的限制可能会影响写入流的操作，因此您应该设置警报阈值来检测此类限制。根据应用程序的限制请求和重试配置，将阈值设置为百分比。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Lambda

ClaimedAccountConcurrency

维度：None

警报描述：此警报有助于监控 Lambda 函数的并发是否接近账户的区域级并发限制。如果函数达到并发限制，它就会开始受到限制。您可以执行下列操作来避免限制。

1. 在此区域请求增加并发。
2. 识别并减少任何未使用的预留并发或预配置的并发。
3. 确定函数中的性能问题，以提高处理速度，从而提高吞吐量。
4. 增加函数的批处理大小，以便每次函数调用都能处理更多消息。

意图：此警报可以主动检测 Lambda 函数的并发是否接近账户的区域级并发限额，以便您可以视情况采取行动。如果 `ClaimedAccountConcurrency` 达到账户的区域级并发限额，则该函数将受到节流。如果您使用的是预留并发（RC）或预调配并发 （PC），则与 `ConcurrentExecutions` 上的警报相比，此警报可让您更清楚地了解并发利用率。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：您应该计算账户在该地区设置的并发配额大约 90% 的值，并将结果用作阈值。默认情况下，您的账户针对区域内所有函数的并发限额为 1000。但是，您应该从“服务配额”控制面板中查看账户的配额。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

错误

维度：FunctionName

警报描述：此警报会检测高错误计数。错误包括代码引发的异常和 Lambda 运行时系统引发的异常。您可以检查与函数相关的日志来诊断问题。

意图：此警报有助于检测函数调用中的高错误计数。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：将阈值设置为大于 0 的数字。确切的值可取决于对您应用程序中的错误的容忍度。了解函数正在处理之调用的临界程度。对于某些应用程序，任何错误都可能是不可接受的，而某些应用程序可能允许一定的错误幅度。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：GREATER_THAN_THRESHOLD

限制

维度：FunctionName

警报描述：此警报会检测大量受限调用请求。限制在无并发可用于纵向扩展时发生。有多种方法可解决此问题。1) 向此区域的 AWS Support 请求增加并发。2) 确定函数中的性能问题，以提高处理速度，从而提高吞吐量。3) 增加函数的批处理大小，以便每次函数调用都能处理更多消息。

意图：此警报有助于检测针对 Lambda 函数的大量受限调用请求。请务必了解请求是否由于限制而持续被拒，以及您是否需要提高 Lambda 函数性能或并发能力以避免持续的限制。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：将阈值设置为大于 0 的数字。阈值的确切值可取决于应用程序的容忍度。根据函数的使用和扩展要求设置阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Duration

维度：FunctionName

警报描述：此警报会检测 Lambda 函数处理事件的长持续时间。持续时间长可能是因为函数代码的变化使函数的执行时间更长，或者函数的依赖项需要更长的时间。

意图：此警报可以检测 Lambda 函数的长运行持续时间。运行时系统持续时间长表示函数的调用时间较长，如果 Lambda 处理的事件数量更多，还会影响调用的并发能力。请务必了解 Lambda 函数的执行时间是否持续比预期的要长。

统计数据：p90

建议阈值：取决于您的情况

阈值理由：持续时间阈值取决于您的应用程序和工作负载以及您的性能要求。对于高性能要求，可以将阈值设置为更短的时间，以查看函数是否符合预期。您还可以分析持续时间指标的历史数据，查看所花费的时间是否与函数的预期性能相符，然后将阈值设置为比历史平均值更长的时间。确保将阈值设置为低于配置的函数超时。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

ConcurrentExecutions

维度：FunctionName

警报描述：此警报有助于监控函数的并发是否接近账户的区域级并发限制。如果函数达到并发限制，它就会开始受到限制。您可以执行下列操作来避免限制。

1. 在此区域请求增加并发。
2. 确定函数中的性能问题，以提高处理速度，从而提高吞吐量。

3. 增加函数的批处理大小，以便每次函数调用都能处理更多消息。

为了更好地了解预留并发和预调配的并发利用率，请改为对新指标 `ClaimedAccountConcurrency` 设置警报。

意图：此警报可以主动检测函数的并发是否接近账户的区域级并发限额，以便您可以视情况采取行动。如果函数达到账户的区域级并发限额，则该函数将受到限制。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：将阈值设置成为区域中账户设置的并发限额的 90% 左右。默认情况下，您的账户针对区域内所有函数的并发限额为 1000。不过，您可以查看账户的限额，因为可以联系 AWS Support 来增加限额。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

Lambda Insights

我们建议为以下 Lambda Insights 指标设置最佳实践警报。

memory_utilization

维度：function_name

警报描述：此警报用于检测 Lambda 函数的内存利用率是否接近配置的限制。要排查问题，您可以尝试 1) 优化您的代码。2) 通过准确估算内存需求来正确调整内存分配的大小。您可以参考 Lambda Power Tuning 了解相同内容。3) 使用连接池。有关 RDS 数据库的连接池的信息，请参阅 Using Amazon RDS Proxy with Lambda。4) 您也可以考虑设计函数，以免两次调用之间在内存中存储大量数据。

意图：此警报用于检测 Lambda 函数的内存利用率是否接近配置的限制。

统计数据：平均值

阈值建议：90.0

阈值理由：将阈值设置为 90%，以便在内存利用率超过分配内存的 90% 时收到提醒。如果您担心工作负载的内存利用率，则可以将此阈值调整为较低的值。您也可以查看此指标的历史数据并相应地设置阈值。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

Amazon VPC (**AWS/NATGateway**)

ErrorPortAllocation

维度：NatGatewayId

警报描述：此警报有助于检测 NAT 网关何时无法为新连接分配端口。要解决此问题，请参阅解决 NAT 网关上的端口分配错误。

意图：此警报用于检测 NAT 网关是否无法分配源端口。

统计数据：Sum

建议阈值：0.0

阈值理由：如果 ErrorPortAllocation 的值大于 0，则意味着通过 NatGateway 打开了太多与单个热门目标的并发连接。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

PacketsDropCount

维度：NatGatewayId

警报描述：此警报有助于检测 NAT 网关何时丢弃数据包。这可能是因为 NAT 网关存在问题，因此请检查 AWS 服务运行状况控制面板，了解区域中的 AWS NAT 网关的状态。这可以帮助您弄清楚与使用 NAT 网关的流量相关的网络问题。

意图：此警报用于检测 NAT 网关是否正丢弃数据包。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：您应计算 NAT 网关上总流量 0.01% 的值，并将该结果用作阈值。使用 NAT 网关上流量的历史数据来确定阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

AWS 私有链接 (**AWS/PrivateLinkEndpoints**)

PacketsDropped

维度：VPC Id、VPC Endpoint Id、Endpoint Type、Subnet Id、Service Name

警报描述：此警报通过监控端点丢弃的数据包数量，帮助检测端点或端点服务是否运行状况不佳。请注意，到达 VPC 端点的大小超过 8500 字节的数据包将被丢弃。要进行问题排查，请参阅接口 VPC 端点和端点服务之间的连接问题。

意图：此警报用于检测端点或端点服务是否运行状况不佳。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：根据用例设置阈值。如果您想了解端点或端点服务的运行状况不佳状态，应将阈值设置得较低，以便有机会在出现大量数据丢失之前修复问题。您可以使用历史数据来了解对丢弃数据包的容忍度，并相应地设置阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

AWS 私有链接 (**AWS/PrivateLinkServices**)

RstPacketsSent

维度：Service Id、Load Balancer Arn、Az

警报描述：此警报有助于您根据发送到端点之重置数据包的数量来检测端点服务的运行状况不佳目标。当您使用服务的消费端调试连接错误时，可以验证服务是否正在使用 rstPacketsSent 指标重置连接，或者网络路径上是否有其他操作会失败。

意图：此警报用于检测端点服务运行状况不佳的目标。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：该阈值取决于用例。如果您的用例可以容忍运行状况不佳目标，则可以将阈值设置得较高。如果用例无法容忍运行状况不佳目标，则可以将阈值设置得非常低。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon RDS

CPU 利用率

维度：DBInstanceIdentifier

警报描述：此警报有助于监控 CPU 利用率持续较高的情况。CPU 利用率衡量非空闲时间。考虑使用增强监控或 Performance Insights 来查看对于 MariaDB、MySQL、Oracle 和 PostgreSQL，哪

些等待时间消耗的 CPU 时间最多（guest、irq、wait、nice 等）。然后评估哪些查询消耗的 CPU 量最高。如果您无法调整工作负载，可以考虑改用更大的数据库实例类。

意图：此警报用于检测 CPU 利用率持续较高的情况，以防响应时间过长和超时。如果要检查 CPU 利用率的微爆，可以设置较短的警报评估时间。

统计数据：平均值

建议阈值：90.0

阈值调整：CPU 消耗的随机峰值可能不会影响数据库性能，但是 CPU 利用率持续较高可能会阻碍即将到来的数据库请求。根据数据库的总体工作负载，您的 RDS/Aurora 实例的 CPU 利用率较高可能会降低整体性能。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

DatabaseConnections

维度：DBInstanceIdentifier

警报描述：此警报可以检测大量连接。检查现有连接并终止任何处于“休眠”状态或未正确关闭的连接。考虑使用连接池来限制新连接的数量。或者，增加数据库实例大小以使用具有更多内存的类，由此增加“max_connections”的默认值，或者如果当前类可以支持您的工作负载，则增加 RDS、Aurora MySQL 和 PostgreSQL 中的“max_connections”值。

意图：当达到最大数据库连接数时，此警报用于帮助防止连接被拒绝。如果您经常更改数据库实例类，则不建议使用此警报，因为这样做会更改内存和默认的最大连接数。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：允许的连接数取决于数据库实例类的大小以及与进程/连接相关的数据库引擎特定参数。您应计算一个介于数据库最大连接数的 90-95% 之间的值，并将该结果用作阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

EBSByteBalance%

维度：DBInstanceIdentifier

警报描述：此警报有助于监控剩余吞吐量积分的低百分比。要进行问题排查，请查看 RDS 中的延迟问题。

意图：此警报用于检测突增存储桶中剩余的低百分比吞吐量积分。低字节余额百分比可能会导致吞吐量瓶颈问题。不建议对 Aurora PostgreSQL 实例使用此警报。

统计数据：平均值

建议阈值：10.0

阈值调整：吞吐量积分余额低于 10% 被视为较差，您应相应地设置阈值。如果您的应用程序可以容忍较低的工作负载吞吐量，也可以设置较低的阈值。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：LESS_THAN_THRESHOLD

EBSIOBalance%

维度：DBInstanceIdentifier

警报描述：此警报有助于监控剩余的 IOPS 积分的低百分比。要进行问题排查，请参阅 RDS 中的延迟问题。

意图：此警报用于检测突增存储桶中剩余的低百分比 I/O 积分。低 IOPS 余额百分比可能会导致 IOPS 瓶颈问题。建议不要对 Aurora 实例使用此警报。

统计数据：平均值

建议阈值：10.0

阈值调整：IOPS 积分余额低于 10% 被视为较差，您可以相应地设置阈值。如果您的应用程序可以容忍较低的工作负载 IOPS，也可以设置较低的阈值。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：LESS_THAN_THRESHOLD

FreeableMemory

维度：DBInstanceIdentifier

警报描述：此警报有助于监控可用内存不足的情况，这可能意味着数据库连接出现峰值或您的实例可能承受较高的内存压力。除了 `FreeableMemory` 之外，还可以通过监控 `SwapUsage` 的 CloudWatch 指标来检查内存压力。如果实例内存消耗频繁过高，这表示您应检查您的工作负载或升级您的实例类。对于 Aurora 读取器数据库实例，请考虑向集群添加额外的读取器数据库实例。有关 Aurora 问题排查的更多信息，请参阅可用内存问题。

意图：此警报有利于防止内存不足的问题，此问题会导致连接被拒绝。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：根据工作负载和实例类，可能适合使用不同的阈值。理想情况下，可用内存不应长时间低于总内存的 25%。对于 Aurora，您可以将阈值设置为接近 5%，因为指标接近 0 意味着数据库实例已经尽可能地扩展。您可以分析该指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：LESS_THAN_THRESHOLD

FreeLocalStorage

维度：DBInstanceIdentifier

警报描述：此警报有助于监控本地可用存储空间不足的问题。Aurora PostgreSQL 兼容版使用本地存储空间来存储错误日志和临时文件。Aurora MySQL 使用本地存储来存储错误日志、一般日志、慢速查询日志、审核日志和非 InnoDB 临时表。这些本地存储卷由 Amazon EBS Store 提供支持，

并可以通过使用更大的数据库实例类来进行扩展。要进行问题排查，请查看 Aurora PostgreSQL 兼容版和 MySQL 兼容版。

意图：此警报用于检测如果您未使用 Aurora Serverless v2 或更高版本，Aurora 数据库实例与本地存储空间限制之间的差距。当您在本地存储空间中存储非永久性数据（例如临时表和日志文件）时，本地存储空间可能会达到容量上限。此警报可以防止在数据库实例用尽本地存储空间时，发生空间不足的错误。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：您应根据卷的使用速度和趋势计算可用存储量的约 10%-20%，然后使用该结果作为阈值，在卷达到限制之前主动采取行动。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

FreeStorageSpace

维度：DBInstanceIdentifier

警报描述：此警报可以监控可用存储空间不足的问题。如果您经常接近存储空间容量限制，请考虑纵向扩展数据库存储。包含一些缓冲区，以适应不可预见的应用程序需求增加。或者，可以考虑启用 RDS 存储空间自动扩缩。此外，可以考虑通过删除未使用或过时的数据和日志来释放更多空间。有关更多信息，请查看 RDS 用尽存储空间文档和 PostgreSQL 存储空间问题文档。

意图：此警报有助于防止存储空间已满的问题。这可以防止数据库实例用尽存储空间时出现停机。如果您启用了存储空间自动扩缩，或者您经常更改数据库实例的存储容量，我们不建议使用此警报。

统计数据：Minimum

建议阈值：取决于您的情况

阈值调整：阈值将取决于当前分配的存储空间。通常，您应计算已分配存储空间的 10% 的值，并将该结果用作阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

MaximumUsedTransactionIDs

维度：DBInstanceIdentifier

警报描述：此警报有助于防止 PostgreSQL 的事务 ID 重叠。请参阅此博客中的问题排查步骤来调查和解决问题。您也可以参考此博客，进一步熟悉 autovacuum 的概念、常见问题和最佳实践。

意图：此警报用于帮助防止 PostgreSQL 的事务 ID 重叠。

统计数据：平均值

建议阈值：1.0E9

阈值调整：将此阈值设置为 10 亿应该可以让您有时间调查问题。默认的 autovacuum_freeze_max_age 值为 2 亿。如果最早的事务的期限为 10 亿，则 autovacuum 很难将该阈值保持在 2 亿个事务 ID 的目标以下。

时间段：60

要发出警报的数据点数：1

评估期：1

比较运算符：GREATER_THAN_THRESHOLD

ReadLatency

维度：DBInstanceIdentifier

警报描述：此警报有助于监控高读取延迟的情况。如果存储延迟很高，那是因为工作负载超过了资源限制。您可以查看相对于实例和分配的存储配置的 I/O 利用率。请参阅排查由 IOPS 瓶颈导致的 Amazon EBS 卷延迟问题。对于 Aurora，您可以切换到具有 I/O 优化存储配置的实例类。有关指导，请参阅 Planning I/O in Aurora。

意图：此警报用于检测高读取延迟。数据库磁盘的读取/写入延迟通常较低，但可能存在导致高延迟操作的问题。

统计数据：p90

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于您的用例。读取延迟高于 20 毫秒时通常都需要调查。如果您的应用程序可能具有更高的读取操作延迟，则也可以设置更高的阈值。查看读取延迟的严重性和要求，并分析此指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

ReplicaLag

维度：DBInstanceIdentifier

警报描述：此警报有助于您了解副本落后于主实例的秒数。如果源数据库实例上未发生任何用户事务，则 PostgreSQL 只读副本将在最迟 5 分钟后报告复制滞后。当 ReplicaLag 指标达到 0 时，即表示副本已赶上主数据库实例进度。如果 ReplicaLag 指标返回 –1，则当前未激活复制。有关与 RDS PostgreSQL 相关的指导，请参阅复制最佳实践；有关 `ReplicaLag` 和相关错误的问题排查，请参阅 ReplicaLag 问题排查。

意图：此警报可以检测副本延迟，这反映了主实例出现故障时可能发生的数据丢失。如果副本落后于主实例太远而主实例出现故障，则副本将丢失主实例中的数据。

统计数据：Maximum

建议阈值：60.0

阈值调整：通常，可接受的延迟取决于应用程序。我们建议不要超过 60 秒。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：GREATER_THAN_THRESHOLD

WriteLatency

维度：DBInstanceIdentifier

警报描述：此警报有助于监控高写入延迟。如果存储延迟很高，那是因为工作负载超过了资源限制。您可以查看相对于实例和分配的存储配置的 I/O 利用率。请参阅排查由 IOPS 瓶颈导致的 Amazon EBS 卷延迟问题。对于 Aurora，您可以切换到具有 I/O 优化存储配置的实例类。有关指导，请参阅 Planning I/O in Aurora。

意图：此警报用于检测高写入延迟。尽管数据库磁盘通常具有较低的读取/写入延迟，但可能会遇到导致高延迟操作的问题。对此进行监控可以确保磁盘延迟与预期一样低。

统计数据：p90

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于您的用例。写入延迟高于 20 毫秒时通常都需要调查。如果您的应用程序可能具有更高的写入操作延迟，则也可以设置更高的阈值。查看写入延迟的严重性和要求，并分析此指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

DBLoad

维度：DBInstanceIdentifier

警报描述：此警报有助于监控高数据库负载。如果进程数超过 vCPU 的数量，则进程开始排队。当队列增加时，性能会受到影响。如果数据库负载经常高于最大 vCPU 并且主要等待状态为 CPU，则表示 CPU 过载。在这种情况下，您可以在“Performance Insights/增强监控”中监控 `CPUUtilization`、`DBLoadCPU` 和排队的任务。您可能需要限制与实例的连接数，优化具有高 CPU 负载的任何 SQL 查询，或考虑使用更大的实例类。如果始终有大量实例处于任何等待状态，则表示可能存在要解决的瓶颈或资源争用问题。

意图：此警报用于检测高数据库负载的情况。高数据库负载可能会导致数据库实例出现性能问题。此警报不适用于无服务器数据库实例。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：最大 vCPU 值由数据库实例的 vCPU（虚拟 CPU）内核数决定。根据最大 vCPU 的不同，可能适合使用不同的阈值。理想情况下，数据库负载不应超过 vCPU 线。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

AuroraVolumeBytesLeftTotal

维度：DBClusterIdentifier

警报描述：此警报有助于监控卷剩余总量过低的情况。当卷剩余总量达到大小限制时，集群会报告空间不足错误。Aurora 存储空间会根据集群卷中的数据自动扩展，并根据数据库引擎版本扩展至 128 TiB 或 64 TiB。考虑通过删除不再需要的表和数据库来减少存储空间。有关更多信息，请参阅存储空间扩展。

意图：此警报用于检测 Aurora 集群与卷大小限制的差距。此警报可以防止在集群用尽空间时，发生空间不足的错误。此警报仅推荐用于 Aurora MySQL。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：您应根据卷使用量增加的速度和趋势计算实际大小限制的 10%-20%，然后使用该结果作为阈值，在卷达到限制之前主动采取行动。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

AuroraBinlogReplicaLag

维度：DBClusterIdentifier、Role=WRITER

警报描述：此警报有助于监控 Aurora 写入器实例复制的错误状态。有关更多信息，请参阅跨AWS区域复制 Aurora MySQL 数据库集群。有关问题排查，请参阅 Aurora MySQL 复制问题。

意图：此警报用于检测写入器实例是否处于错误状态并且无法复制源。此警报仅推荐用于 Aurora MySQL。

统计数据：平均值

建议阈值：–1.0

阈值调整：我们建议您使用 -1 作为阈值，因为如果副本处于错误状态，Aurora MySQL 会发布此值。

时间段：60

要发出警报的数据点数：2

评估期：2

比较运算符：LESS_THAN_OR_EQUAL_TO_THRESHOLD

BlockedTransactions

维度：DBInstanceIdentifier

警报描述：此警报有助于监控 Aurora 数据库实例中被阻止事务数量高的情况。被阻止的事务最后可以回滚或提交。高并发度、事务空闲或事务长时间运行都可能导致事务被阻止。有关问题排查，请参阅 Aurora MySQL 文档。

意图：此警报用于检测 Aurora 数据库实例中被阻止事务数量高的情况，以防止事务回滚和性能下降。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：您应使用 `ActiveTransactions` 指标计算实例所有事务的 5%，并将该结果用作阈值。您还可以查看被阻止事务的严重性和要求，并分析此指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

BufferCacheHitRatio

维度：DBInstanceIdentifier

警报描述：此警报有助于您监控 Aurora 集群缓存命中率持续较低的情况。低命中率表示您对这个数据库实例的查询经常转向磁盘。有关问题排查，请调查您的工作负载以查看哪些查询导致了此行为，并参阅数据库实例 RAM 建议文档。

意图：此警报用于检测缓存命中率持续较低的情况，以防 Aurora 实例的性能持续下降。

统计数据：平均值

建议阈值：80.0

阈值调整：您可以将缓冲区缓存命中率的阈值设置为 80%。但是，您可以根据可接受性能级别和工作负载特征来调整此值。

时间段：60

要发出警报的数据点数：10

评估期：10

比较运算符：LESS_THAN_THRESHOLD

EngineUptime

维度：DBClusterIdentifier、Role=WRITER

警报描述：此警报有助于监控写入器数据库实例停机时间较少的情况。由于重启、维护、升级或失效转移，写入器数据库实例可能会停机。当正常运行时间因集群中的失效转移而达到 0，并且集群具有一个或多个 Aurora 副本时，Aurora 副本会在故障事件期间提升至主写入器实例。要提高数据库集群的可用性，请考虑在两个或更多不同可用区中创建一个或多个 Aurora 副本。有关更多信息，请查看影响 Aurora 停机时间的因素。

意图：此警报用于检测 Aurora 写入器数据库实例是否处于停机状态。这可以防止由于崩溃或失效转移而导致在写入器实例中出现长时间运行的故障。

统计数据：平均值

建议阈值：0.0

阈值调整：故障事件将导致短暂中断，其间的读取和写入操作将失败并引发异常。不过，服务通常会在 60 秒内（经常在 30 秒内）还原。

时间段：60

要发出警报的数据点数：2

评估期：2

比较运算符：LESS_THAN_OR_EQUAL_TO_THRESHOLD

RollbackSegmentHistoryListLength

维度：DBInstanceIdentifier

警报描述：此警报有助于监控 Aurora 实例回滚分段历史记录长度持续较长。InnoDB 历史记录列表长度较长则表明大量的旧行版本、查询和数据库关闭已经变慢。有关更多信息和问题排查，请参阅 InnoDB 历史记录列表长度显著增加文档。

意图：此警报用于检测回滚段历史记录长度持续较长。这有助于您防止 Aurora 实例的持续性能下降和 CPU 利用率过高。此警报仅推荐用于 Aurora MySQL。

统计数据：平均值

建议阈值：1000000.0

阈值调整：将此阈值设置为 100 万可让您有充足的时间调查问题。但是，您可以根据可接受性能级别和工作负载特征来调整此值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

StorageNetworkThroughput

维度：DBClusterIdentifier、Role=WRITER

警报描述：此警报有助于监控存储网络吞吐量较高的情况。如果存储网络吞吐量超过 EC2 实例的总网络带宽，则可能导致读取和写入延迟较高，从而导致性能下降。您可以从 AWS 控制台查看您的

EC2 实例类型。有关问题排查，请检查写入/读取延迟的任何变化，并评估您是否也针对此指标发出了警报。如果是这样的话，请评估警报触发期间的工作负载模式。这有助于您确定是否可以优化工作负载以减少总网络流量。如果无法优化，您可能需要考虑扩展实例。

意图：此警报用于检测存储网络吞吐量较高的情况。检测高吞吐量可以防止网络数据包丢失和性能下降。

统计数据：平均值

建议阈值：取决于您的情况

阈值调整：您应计算出 EC2 实例类型总网络带宽的约 80%-90%，然后使用该结果作为阈值，以便在网络数据包受到影响之前主动采取行动。您还可以查看存储网络吞吐量的严重性和要求，并分析此指标的历史行为以确定合理的阈值级别。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon Route 53 Public Data Plane

HealthCheckStatus

维度：HealthCheckId

警报描述：此警报有助于根据运行状况检查程序检测运行状况不佳的端点。要了解导致运行状况不佳状态的失败的原因，请使用 Route 53 运行状况检查控制台中的“运行状况检查程序”选项卡，查看每个区域的状态以及上次运行状况检查的失败情况。“状态”选项卡还显示端点被报告为运行状况不佳的原因。请参阅问题排查步骤。

意图：此警报使用 Route53 运行状况检查程序来检测运状况不佳的端点。

统计数据：平均值

建议阈值：1.0

阈值理由：当端点运行正常时，其状态报告为 1。所有小于 1 的情况都表示运行状况不佳。

时间段：60

要发出警报的数据点数：3

评估期：3

比较运算符：LESS_THAN_THRESHOLD

Amazon S3

4xxErrors

维度：BucketName、FilterId

警报描述：此警报可帮助我们报告为响应客户端请求而发出的 4xx 错误状态代码的总数。例如，403 错误代码可能表示 IAM 策略不正确，404 错误代码可能表示客户端应用程序操作不当。临时启用 S3 服务器访问日志记录将帮助您使用 HTTP 状态和错误代码字段查明问题的根源。要了解有关错误代码的更多信息，请参阅 Error Responses。

意图：此告警可用于为典型的 4xx 错误率创建基准，以便您可以调查可能表明存在设置问题的任何异常。

统计数据：平均值

建议阈值：0.05

阈值理由：建议阈值旨在检测是否总请求数中超过 5% 的请求会收到 4XX 错误。应针对频繁发生的 4XX 错误设置告警。但是，将阈值设置得非常低可能会导致告警过于敏感。您还可以调整阈值以适应请求的负载，同时考虑可接受的 4XX 错误级别。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

5xxErrors

维度：BucketName、FilterId

警报描述：此警报有助于您检测高客户端错误数。此类错误表明客户端发出了服务器无法完成的请求。这可以帮助您弄清楚您的应用程序因为 S3 而面临的问题。有关帮助您高效处理或减少错误的更多信息，请参阅优化性能设计模式。错误也可能是由 S3 的问题所致，请查看 AWS 服务运行状况控制面板，了解区域中的 Amazon S3 的状态。

意图：此警报有助于检测应用程序是否因 5xx 错误而遇到问题。

统计数据：平均值

建议阈值：0.05

阈值理由：我们建议设置阈值，以检测是否总请求数中超过 5% 的请求会收到 5XX 错误。但是，您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

OperationsFailedReplication

维度：SourceBucket、DestinationBucket、RuleId

警报描述：此警报有助于了解复制失败。此指标会跟踪使用 S3 CRR 或 S3 SRR 复制的新对象的状态，还会跟踪使用 S3 批量复制复制的现有对象。有关更多详细信息，请参阅对复制进行问题排查。

意图：此警报用于检测是否存在失败的复制操作。

统计数据：Maximum

建议阈值：0.0

阈值理由：对于成功的操作，此指标会发出值 0；当一分钟内没有执行任何复制操作时，此指标不发出任何值。当指标发出的值大于 0 时，复制操作将失败。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

S3ObjectLambda

4xxErrors

维度：AccessPointName、DataSourceARN

警报描述：此警报可帮助我们报告为响应客户端请求而发出的 4xx 错误状态代码的总数。临时启用 S3 服务器访问日志记录将帮助您使用 HTTP 状态和错误代码字段查明问题的根源。

意图：此告警可用于为典型的 4xx 错误率创建基准，以便您可以调查可能表明存在设置问题的任何异常。

统计数据：平均值

建议阈值：0.05

阈值理由：我们建议设置阈值，以检测是否总请求数中超过 5% 的请求会收到 4xx 错误。应针对频繁发生的 4XX 错误设置告警。但是，将阈值设置得非常低可能会导致告警过于敏感。您还可以调整阈值以适应请求的负载，同时考虑可接受的 4XX 错误级别。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

5xxErrors

维度：AccessPointName、DataSourceARN

警报描述：此警报有助于检测高客户端错误数。此类错误表明客户端发出了服务器无法完成的请求。此类错误也可能是由 S3 的问题所致，请查看 AWS 服务运行状况控制面板，了解区域中的 Amazon S3 的状态。这可以帮助您弄清楚您的应用程序因为 S3 而面临的问题。有关帮助您高效处理或减少此类错误的信息，请参阅优化性能设计模式。

意图：此警报有助于检测应用程序是否因 5xx 错误而遇到问题。

统计数据：平均值

建议阈值：0.05

阈值理由：我们建议设置阈值，以检测是否总请求数中超过 5% 的请求会收到 5XX 错误。但是，您可以调整阈值以适应请求的流量以及可接受的错误率。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

LambdaResponse4xx

维度：AccessPointName、DataSourceARN

警报描述：此警报有助于您检测和诊断 S3 对象 Lambda 调用中的失败（500 秒）。此类错误可能是由负责响应您请求的 Lambda 函数中的错误或配置错误所致。调查与对象 Lambda 接入点关联的 Lambda 函数的 CloudWatch 日志流，可以帮助您根据 S3 对象 Lambda 的响应查明问题的根源。

意图：此警报用于检测 WriteGetObjectResponse 调用的 4xx 客户端错误。

统计数据：平均值

建议阈值：0.05

阈值理由：我们建议设置阈值，以检测是否总请求数中超过 5% 的请求会收到 4xx 错误。应针对频繁发生的 4XX 错误设置告警。但是，将阈值设置得非常低可能会导致告警过于敏感。您还可以调整阈值以适应请求的负载，同时考虑可接受的 4XX 错误级别。您还可以分析历史数据以确定应用程序工作负载的可接受错误率，然后相应地调整阈值。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_THRESHOLD

Amazon SNS

NumberOfMessagesPublished

维度：TopicName

警报描述：此警报可以检测何时发布的 SNS 消息数量过低。要排查问题，请检查发布者发送的流量减少的原因。

意图：此警报有助于您主动监控和检测通知发布的显著下降。这可以帮助您识别应用程序或业务流程的潜在问题，以便您可以执行适当的操作来维护预期的通知流。如果您希望系统处理的流量最低，则应创建此警报。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：发布的消息数量应与应用程序的预期已发布消息数量一致。您还可以分析历史数据、趋势和流量，以确定合适的阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

NumberOfNotificationsDelivered

维度：TopicName

警报描述：此警报可以检测何时传送的 SNS 消息数量过低。这可能是由于端点意外取消订阅或导致消息出现延迟的 SNS 事件所致。

意图：此警报有助于您检测传送的消息量的下降。如果您希望系统处理的流量最低，则应创建此警报。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：传送的消息数量应与预期生成的消息数量和消费端数量相符。您还可以分析历史数据、趋势和流量，以确定合适的阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：LESS_THAN_THRESHOLD

NumberOfNotificationsFailed

维度：TopicName

警报描述：此警报可以检测何时传送失败的 SNS 消息数量过高。要排查失败通知的问题，请启用日志记录功能，并将日志记录到 CloudWatch Logs 中。检查日志可以帮助您确定面向哪些订阅用户的通知失败了，以及这些通知返回的状态码。

意图：此警报有助于您主动发现通知传送方面的问题，并执行适当的操作来解决这些问题。

统计数据：Sum

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于失败通知的影响。查看提供给最终用户的 SLA、通知的容错能力和临界程度，并分析历史数据，然后相应地选择阈值。对于只有 SQS、Lambda 或 Firehose 订阅的主题，失败的通知数量应为 0。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidAttributes

维度：TopicName

警报描述：此警报有助于监控和解决发布者或订阅用户遇到的潜在问题。检查发布者是否在发布具有无效属性的消息，或者是否对订阅用户应用了不当筛选条件。您还可以分析 CloudWatch Logs，以帮助确定问题的根本原因。

意图：此警报用于检测已发布的消息是否无效，或者是否对订阅用户应用了不当筛选条件。

统计数据：Sum

建议阈值：0.0

阈值理由：无效属性几乎总是发布者出错导致的。我们建议将阈值设置为 0，因为在运行正常的系统中，预计不会出现无效属性。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidMessageBody

维度：TopicName

警报描述：此警报有助于监控和解决发布者或订阅用户遇到的潜在问题。检查发布者是否在发布具有无效消息正文的消息，或者是否对订阅用户应用了不当筛选条件。您还可以分析 CloudWatch Logs，以帮助确定问题的根本原因。

意图：此警报用于检测已发布的消息是否无效，或者是否对订阅用户应用了不当筛选条件。

统计数据：Sum

建议阈值：0.0

阈值理由：无效消息正文几乎总是发布者出错导致的。我们建议将阈值设置为 0，因为在运行正常的系统中，预计不会出现无效消息正文。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

NumberOfNotificationsRedrivenToDlq

维度：TopicName

警报描述：此警报有助于监控要移至死信队列的消息数量。

意图：此警报用于检测要移至死信队列的消息。我们建议您在将 SNS 与 SQS、Lambda 或 Firehose 结合使用时创建此警报。

统计数据：Sum

建议阈值：0.0

阈值理由：在任何订阅用户类型的运行正常的系统中，不应将消息移至死信队列。如果有任何消息进入该队列，我们建议向您发送通知，以便您可以识别根本原因并解决问题，并有可能重新驱动死信队列中的消息以防止数据丢失。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

NumberOfNotificationsFailedToRedriveToDlq

维度：TopicName

警报描述：此警报有助于监控无法移至死信队列的消息。检查您的死信队列是否存在，并检查其配置是否正确。此外，请验证 SNS 是否有权访问死信队列。要了解更多信息，请参阅死信队列文档。

意图：此警报用于检测无法移至死信队列的消息。

统计数据：Sum

建议阈值：0.0

阈值理由：如果无法将消息移至死信队列，则几乎总会出错。建议阈值为 0，这意味着在配置队列后，所有处理失败的消息都必须能够到达死信队列。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

SMSMonthToDateSpentUSD

维度：TopicName

警报描述：此警报有助于监控您的账户中是否有足够的限额让 SNS 能够传送消息。如果达到了限额，SNS 将无法传送 SMS 消息。有关设置您的每月 SMS 支出限额的信息，或有关向 AWS 请求提高支出限额的信息，请参阅设置发送 SMS 消息的首选项。

意图：此警报用于检测您的账户中是否有足够的限额来成功传送 SMS 消息。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：根据账户的限额（账户支出限额）设置阈值。选择一个阈值，该阈值会尽早告知您已达到限额，以便您有时间请求增加限额。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

SMSSuccessRate

维度：TopicName

警报描述：此警报有助于监控 SMS 消息传送的失败率。您可以设置 Cloudwatch Logs 以了解失败的性质并据此执行操作。

意图：此警报用于检测 SMS 消息传送失败。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：根据您对 SMS 消息传送失败的容忍度设置警报阈值。

时间段：60

要发出警报的数据点数：5

评估期：5

比较运算符：GREATER_THAN_THRESHOLD

Amazon SQS

ApproximateAgeOfOldestMessage

维度：QueueName

警报描述：此警报会监控队列中最早消息的期限。您可以使用此警报来监控您的消费端是否以所需速度处理 SQS 消息。考虑增加消费端数量或消费端吞吐量以缩短消息期限。此指标可以与 `ApproximateNumberOfMessagesVisible` 结合使用，以确定队列积压的大小以及消息的处理速度。为防止消息在处理之前被删除，请考虑将死信队列配置为放弃潜在的毒丸消息。

意图：此警报用于检测 queueName 队列中最早消息的期限是否过长。长期限可能表示消息处理速度不够快，或者有一些毒丸消息卡在队列中无法处理。

统计数据：Maximum

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于预期的消息处理时间。您可以使用历史数据计算平均消息处理时间，然后将阈值设置为比队列消费端的预期最大 SQS 消息处理时间多 50%。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesNotVisible

维度：QueueName

警报描述：此警报有助于检测与 `QueueName` 有关的大量传输中消息。要排查问题，请检查消息积压减少。

意图：此警报用于检测队列中大量的传输中消息。如果消费端没有在可见性超时期限内删除消息，则在轮询队列时，消息会重新出现在队列中。对于 FIFO 队列，最多可以有 20000 条传输中消息。如果您达到此限额，SQS 将不会返回任何错误消息。FIFO 队列会浏览前 2 万条消息以确定可用的消息组。这意味着，如果您在单个消息组中积压了消息，则在成功使用积压的消息之前，您无法使用其他消息组中稍后发送到该队列的消息。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：此警报的建议阈值在很大程度上取决于预期的传输中消息数量。您可以使用历史数据来计算传输中消息的预期最大数量，并将阈值设置为超过该值的 50%。如果队列的消费端正在处理消息但没有从队列中删除消息，则此数量将突然增加。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesVisible

维度：QueueName

警报描述：此警报会监控消息队列的积压是否大于预期，这表明消费端的速度太慢或消费端数量不足。如果此警报进入“ALARM”状态，可以考虑增加消费端数量或加快消费端的速度。

意图：此警报用于检测活动队列的消息计数是否过高，消费端是否处理消息缓慢，或者是否没有足够的消费端来处理消息。

统计数据：平均值

建议阈值：取决于您的情况

阈值理由：可见消息的数量异常高，表明消费端处理消息的速度未达到预期。设置此阈值时，应考虑历史数据。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：GREATER_THAN_OR_EQUAL_TO_THRESHOLD

NumberOfMessagesSent

维度：QueueName

警报描述：此警报有助于检测生成器是否没有就 QueueName 发送任何消息。要排查问题，请检查生成器未发送消息的原因。

意图：此警报用于检测生成器何时停止发送消息。

统计数据：Sum

建议阈值：0.0

阈值理由：如果发送的消息数量为 0，则生成器不会发送任何消息。如果此队列的 TPS 较低，请相应地增加 EvaluationPeriods 的数量。

时间段：60

要发出警报的数据点数：15

评估期：15

比较运算符：LESS_THAN_OR_EQUAL_TO_THRESHOLD

AWS VPN

TunnelState

维度：VpnId

警报描述：此警报有助于您了解一条或多条隧道的状态是否为“DOWN”。有关问题排查，请参阅如何排查与 Amazon VPC 的 VPN 隧道连接故障？。

意图：此警报用于检测此 VPN 的至少一条隧道是否处于“DOWN”状态，以便您可以排查受影响 VPN 的问题。对于仅配置了单条隧道的网络，此警报将始终处于“ALARM”状态。

统计数据：Minimum

建议阈值：1.0

阈值理由：值小于 1 表示至少有一条隧道处于“DOWN”状态。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：LESS_THAN_THRESHOLD

TunnelState

维度：TunnelIpAddress

警报描述：此警报有助于您了解此隧道的状态是否为“DOWN”。有关问题排查，请参阅如何排查与 Amazon VPC 的 VPN 隧道连接故障？。

意图：此警报用于检测隧道是否处于“DOWN”状态，以便您可以排查受影响 VPN 的问题。对于仅配置了单条隧道的网络，此警报将始终处于“ALARM”状态。

统计数据：Minimum

建议阈值：1.0

阈值理由：值小于 1 表示隧道处于“DOWN”状态。

时间段：300

要发出警报的数据点数：3

评估期：3

比较运算符：LESS_THAN_THRESHOLD

根据指标触发警报

以下各节中的步骤说明了如何创建监测指标的 CloudWatch 警报。

根据静态阈值创建 CloudWatch 告警

您可以为告警选择一个要监视的 CloudWatch 指标，并为该指标选择阈值。如果该指标在指定数量的评估期内超出阈值，告警将变为 ALARM（告警）状态。

如果您创建告警的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则可以设置告警以监测与该监控账户关联的源账户中的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

根据单个指标创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。

3. 选择创建警报。
4. 选择 Select Metric（选择指标）。
5. 请执行以下操作之一：
   - 选择包含所需指标的服务命名空间。继续选择所显示的选项，以缩小选择范围。在显示指标列表时，选中所需的指标旁边的复选框。
   - 在搜索框中，输入指标名称、账户 ID、账户标签、维度或资源 ID。接下来，选择其中的一个结果并继续，直到显示指标列表。选中所需的指标旁边的复选框。
6. 选择 Graphed metrics（绘制的指标）选项卡。
   a. 在 Statistic（统计数据）下，选择其中的一个统计数据或预定义百分比值，或者指定一个自定义百分比值（例如 **`p95.45`**）。
   b. 在 Period（时间段）下，选择告警的评估期。评估告警时，每个时间段都聚合到一个数据点。

      在创建告警时，您还可以选择是在左侧还是右侧显示 Y 轴图例。该首选项仅在创建告警时使用。
   c. 选择选择指标。

      将显示 Specify metric and conditions（指定指标和条件）页面，其中显示一个图表以及有关您选择的指标和统计数据的其他信息。
7. 在条件下面，指定以下内容：
   a. 对于 Whenever ***`metric`*** is（每当指标），指定指标是必须大于、小于还是等于阈值。在于... 下面，指定阈值。
   b. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

      要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。
   c. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
   d. 如果警报将百分比值作为监控的统计数据，将显示样本数少的百分比框。使用它来选择是评估还是忽略采样率低的案例。如果选择忽略 (保持警报状态)，在样本大小太小时，将始终保持当前警报状态。有关更多信息，请参阅 基于百分位数的 CloudWatch 告警和小数据样本。
8. 选择下一步。

9. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

   要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

   在 CloudWatch 跨账户可观测性中，您可以选择向多个 AWS 账户发送通知。例如，同时向监控账户和源账户发送通知。

   要让警报不发送通知，请选择删除。

10. 要让警报执行 Auto Scaling、EC2、Lambda 或 Systems Manager 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。

11. 在完成后，选择下一步。
12. 输入警报的名称和说明。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。然后选择下一步。
13. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

您还可以将警报添加到控制面板。有关更多信息，请参阅 在 CloudWatch 控制面板上添加或删除警报小组件 。

根据指标数学表达式创建 CloudWatch 告警

要基于指标数学表达式创建告警，请选择表达式中要使用的一个或多个 CloudWatch 指标。然后，指定表达式、阈值和评估期。

无法创建基于 SEARCH（搜索）表达式的告警。这是因为搜索表达式可返回多个时间序列，而基于数学表达式的告警只能监视一个时间序列。

根据指标数学表达式创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Alarms（告警），然后选择 All alarms（所有告警）。
3. 选择创建警报。
4. 选择 Select Metric（选择指标），然后执行以下操作之一:
   - 从 AWS namespaces（命名空间）下拉列表或 Custom namespaces（自定义命名空间）下拉列表中，选择命名空间。选择命名空间后，您可以继续选择选项，直到显示指标列表，您可以在其中选择正确指标旁边的复选框。
   - 使用搜索框查找指标、账户 ID、维度或资源 ID。输入指标、维度或资源 ID 后，您可以继续选择选项，直到显示指标列表，您可以在其中选择正确指标旁边的复选框。
5. （可选）如果要向指标数学表达式添加另一个指标，您可以使用搜索框查找特定指标。您可以向指标数学表达式添加多达 10 个指标。
6. 选择 Graphed metrics（绘制的指标）选项卡。对于您之前添加的每个指标，请执行以下操作:

   a. 在 Statistic（统计数据）列中，选择下拉菜单。在下拉菜单中，选择一个预定义的统计数据或百分位数。使用下拉菜单中的搜索框指定自定义百分位数。

   b. 在 Period（时间段）列下，选择下拉菜单。在下拉菜单中，选择一个预定义的评估期。

   在创建告警时，您可以指定 Y 轴图例是在左侧还是右侧显示。

   > **Note**
   >
   > 在 CloudWatch 评估告警时，时间段会聚合到单个数据点中。

7. 选择 Add math（添加数学）下拉菜单，然后从预定义的指标数学表达式列表中选择 Start with an empty expression（从空表达式开始）。

   选择 Start with an empty expression（从空表达式开始）之后，系统将显示一个数学表达式框，您可以在其中应用或编辑数学表达式。
8. 在数学表达式框中，输入数学表达式，然后选择 Apply（应用）。

   选择 Apply（应用）之后，Label（标注）列旁边会显示一个 ID 列。

   要将一个指标或另一个指标数学表达式的结果用作当前数学表达式公式的一部分，您可以使用 ID 列下显示的值。要更改 ID 值，您可以选择当前值旁边的笔和纸图标。新值必须以小写字母开头，并且可以包含数字、字母和下划线符号。将 ID 值更改为更明显的名称也可以使告警图表更易于理解。

   有关可用于指标数学的函数的信息，请参阅 指标数学语法和函数。

9. （可选）在新数学表达式的公式中，使用指标和其他数学表达式的结果添加更多数学表达式。
10. 当您具有要用于告警的表达式时，请清除页面上每个其他表达式和每个指标左侧的复选框。只应选中要用于告警的表达式旁边的复选框。您为告警选择的表达式必须生成单个时间序列，并且仅在图表上显示一行。然后选择 Select metric（选择指标）。

    将显示 Specify metric and conditions（指定指标和条件）页面，其中显示一个图表以及有关您选择的数学表达式的其他信息。
11. 对于 Whenever ***expression*** is（每当表达式），指定表达式是必须大于、小于还是等于阈值。在于... 下面，指定阈值。
12. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

    要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。
13. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
14. 选择下一步。
15. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

    要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

    要让警报不发送通知，请选择删除。
16. 要让警报执行 Auto Scaling、EC2、Lambda 或 Systems Manager 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。如果您选择 Lambda 函数作为警报操作，则需要指定函数名称或 ARN，并且可以选择该函数的特定版本。

    告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。
17. 在完成后，选择下一步。

18. 输入警报的名称和说明。然后选择下一步。

    名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。

19. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

您还可以将警报添加到控制面板。有关更多信息，请参阅 在 CloudWatch 控制面板上添加或删除警报小组件 。

基于 Metrics Insights 查询创建 CloudWatch 告警

您可以对返回单个时间序列的任何 Metrics Insights 查询创建告警。这对于创建动态告警以监视基础设施或应用程序实例集的聚合指标特别有用。只需创建一次告警，它就会随着实例集添加或删除资源而调整。例如，您可以创建一个监视所有实例的 CPU 利用率的告警，该告警会随着您添加或删除实例而动态调整。

有关完整说明，请参阅为 Metrics Insights 查询创建告警。

基于连接的数据来源创建警报

您可以创建警报，监控源自非 CloudWatch 中的数据来源的指标。有关创建与这些其他数据来源的连接的更多信息，请参阅查询源自其他数据来源的指标。

针对您已连接的数据来源的指标创建警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Metrics（指标）、All metrics（所有指标）。
3. 选择多来源查询选项卡。
4. 对于数据来源，选择要使用的数据来源。
5. 查询生成器会提示您输入查询所需的信息，以检索用于警报的指标。每个数据来源的工作流程都不同，并且这些工作流程都是针对数据来源量身定制的。例如，对于 Amazon Managed Service for Prometheus 和 Prometheus 数据来源，会出现一个带有查询助手的 PromQL 查询编辑器框。
6. 完成查询构造后，选择图表查询。
7. 如果样本图表看起来像您所期望的那样，请选择创建警报。
8. 出现指定指标和条件页面。如果您使用的查询生成多个时间序列，您会在页面顶部看到一个警告横幅。如果这样做，请在聚合函数中选择一个用于聚合时间序列的函数。

9. （可选）为警报添加标签。
10. 对于每当 ***your-metric-name*** 为……，选择大于、大于/等于、小于/等于或小于。然后，对于相比……，为您的阈值指定一个数值。
11. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

    要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。
12. 对于 Missing data treatment（缺失数据处理），选择在缺失某些数据点时的告警行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
13. 选择下一步。
14. 对于通知，选择当您的警报转换为 ALARM、OK 或 INSUFFICIENT_DATA 状态时要通知的 Amazon SNS 主题。

    a. （可选）要为相同告警状态或不同告警状态发送多个通知，请选择 Add notification（添加通知）。

    > **Note**
    >
    > 我们建议您将警报设置为在进入数据不足状态以及进入警报状态时采取行动。这是因为连接到数据来源的 Lambda 函数的许多问题都可能导致警报转换为数据不足。

    b. （可选）如果无需发送 Amazon SNS 通知，请选择移除。
15. 要让警报执行 Auto Scaling、EC2、Lambda 或 Systems Manager 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。如果您选择 Lambda 函数作为警报操作，则需要指定函数名称或 ARN，并且可以选择该函数的特定版本。

    告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。
16. 选择下一步。

17. 在 Name and description（名称和描述）下，输入告警的名称和描述，然后选择 Next（下一步）。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。

> **Tip**
>
> 警报名称只能包含 UTF-8 字符。不能包含 ASCII 控制字符。

18. 在 Preview and create（预览和创建）下，确认告警的信息和条件正确，然后选择 Create alarm（创建告警）。

有关已连接数据来源警报的详细信息

- 当 CloudWatch 评估警报时，即使警报的时间长于一分钟，也会每分钟评估一次。要使警报起作用，Lambda 函数必须能够返回从任何一分钟开始的时间戳列表，而不仅仅是周期长度的倍数。这些时间戳必须相隔一个周期长度。

  因此，如果 Lambda 查询的数据来源只能返回周期长度的倍数的时间戳，则该函数应对获取的数据“重新采样”以匹配 `GetMetricData` 请求所期望的时间戳。

  例如，每分钟评估一个周期为五分钟的警报，使用五分钟的窗口，每次偏移一分钟。在本例中：

  - 对于在 12:15:00 进行的警报评估，CloudWatch 预计数据点的时间戳为 `12:00:00`、`12:05:00` 和 `12:10:00`。
  - 然后，对于在 12:16:00 进行的警报评估，CloudWatch 预计数据点的时间戳为 `12:01:00`、`12:06:00` 和 `12:11:00`。
- 当 CloudWatch 评估警报时，Lambda 函数返回的与预期时间戳不一致的任何数据点都将被丢弃，并使用剩余的预期数据点对警报进行评估。例如，当在 `12:15:00` 对警报进行评估时，其期望的数据的时间戳为 `12:00:00`、`12:05:00` 和 `12:10:00`。如果收到时间戳为 `12:00:00`、`12:05:00`、`12:06:00` 和 `12:10:00` 的数据，则 `12:06:00` 的数据将被丢弃，CloudWatch 会使用其他时间戳评估警报。

  然后，在 `12:16:00` 的下一次评估中，其期望的数据的时间戳为 `12:01:00`、`12:06:00` 和 `12:11:00`。如果只有时间戳为 `12:00:00`、`12:05:00` 和 `12:10:00` 的数据，则所有这些数据点都将在 12:16:00 被忽略，警报会根据您指定的警报处理缺失数据的方式转换状态。有关更多信息，请参阅 评估告警。

- 我们建议您创建这些警报，以便在其转换为 INSUFFICIENT_DATA 状态时采取行动，因为无论您设置警报以何种方式处理缺失数据，多个 Lambda 函数失败用例都会将警报转换为 INSUFFICIENT_DATA。
- 如果 Lambda 函数返回错误或返回部分数据：
  - 如果在调用 Lambda 函数时出现权限问题，则警报会根据您在创建警报时指定的缺失数据处理方式，开始进行缺失数据转换。
  - 如果 Lambda 函数返回 'StatusCode' = 'PartialData'，则警报评估失败，警报将在尝试三次后转换为 INSUFFICIENT_DATA，这大约需要三分钟。
  - 任何其他来自 Lambda 函数的错误都会导致警报转换为 INSUFFICIENT_DATA。
- 如果 Lambda 函数请求的指标有一定的延迟，以至于最后一个数据点总是缺失，则应采用相应的解决方法。您可以创建 M 个警报（共 N 个）或延长警报的评估周期。有关 M 个警报（共 N 个）的更多信息，请参阅 评估告警。

根据异常检测创建 CloudWatch 告警

您可以根据 CloudWatch 异常检测来创建告警，该异常检测可以分析过去的指标数据并创建预期值模型。预期值会考虑指标中的典型每小时、每日和每周模式。

您需要为异常检测阈值设置一个值，然后 CloudWatch 在模型中使用该阈值来确定指标值的“正常”范围。阈值越高，所产生的“正常”值的范围越大。

您可以选择当指标值高于预期值范围、低于预期值范围，或出现二者情况之一时是否触发告警。

您还可以针对单个指标和指标数学表达式的输出创建异常检测告警。您可以使用这些表达式来创建能可视化异常检测范围的图表。

在设置为 CloudWatch 跨账户可观测性的账户中，除了监控账户中的指标外，您还可以在源账户的指标中创建异常检测器。

有关更多信息，请参阅 使用 CloudWatch 异常检测。

> **Note**
>
> 如果您出于可视化目的已在 Metrics（指标）控制台中对某个指标使用了异常检测，并针对该指标创建了异常检测告警，则您为该告警设置的阈值不会更改您为可视化设置的阈值。有关更多信息，请参阅 创建图表。

创建基于异常检测的告警

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。
3. 选择创建警报。
4. 选择 Select Metric（选择指标）。
5. 请执行以下操作之一：
   - 选择包含您的指标的服务命名空间，然后继续选择相应选项，因为这些选项看起来会缩小您的选项范围。在显示指标列表时，选中您的指标旁边的复选框。
   - 在搜索框中，输入指标名称、维度或资源 ID。选择其中的一个结果，然后继续在选项显示时选择它们，直到显示指标列表为止。选中您的指标旁边的复选框。
6. 选择绘制的指标。
   a. （可选）对于统计数据，选择下拉列表，然后选择一个预定义的统计数据或百分位数。您可以使用下拉列表中的搜索框指定自定义百分位数，如 **`p95.45`**。
   b. （可选）对于周期，选择下拉列表，然后选择一个预定义的评估周期。

   > **Note**
   > 当 CloudWatch 评估您的告警时，它会将该周期汇总为单个数据点。对于异常检测告警，评估周期必须是一分钟或更长时间。

7. 选择下一步。
8. 在条件下面，指定以下内容：
   a. 选择 Anomaly detection（异常检测）。

   如果针对此指标和统计数据的模型已存在，则 CloudWatch 在屏幕顶部的图中显示异常检测范围的预览。创建警报后，实际异常检测范围出现在图中可能需要 15 分钟。在此之前，您看到的范围是异常检测范围的近似值。

   > **Tip**
   > 要在更长的时间范围内查看屏幕顶部的图表，请选择屏幕右上方的 Edit（编辑）。

如果此指标和统计的模型不存在，CloudWatch 在您完成创建警报后生成异常检测范围。对于新模型，实际异常检测范围出现在图表中可能需要 3 个小时。新模型的训练可能需要两周时间，因此异常检测范围可以显示更准确的预期值。

b. 对于 Whenever ***metric*** is（每当指标为），指定何时触发告警。例如，每当指标大于、小于或超出范围（在任一方向）。

c. 对于 Anomaly detection threshold（异常检测阈值），选择用于异常检测阈值的数字。数字越大，创建的“正常”值的范围就越大，也更能容忍指标变化。数字越小，创建的范围就越小，它将以较小的指标偏差进入 ALARM 状态。数字不一定是整数。

d. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。

e. 对于 Missing data treatment（缺失数据处理），选择在缺失某些数据点时的告警行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。

f. 如果警报将百分比值作为监控的统计数据，将显示样本数少的百分比框。使用它来选择是评估还是忽略采样率低的案例。如果选择 Ignore（maintain alarm state）（忽略（保持警报状态）），在样本大小太小时，将始终保持当前警报状态。有关更多信息，请参阅 基于百分位数的 CloudWatch 告警和小数据样本。

9. 选择下一步。

10. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

要为相同告警状态或不同告警状态发送多个通知，请选择 Add notification（添加通知）。

如果不想告警发送通知，请选择 Remove（移除）。

11. 您可以将警报设置为在发生状态更改时执行 EC2 操作或调用 Lambda 函数，或在进入 ALARM 状态时创建 Systems Manager OpsItem 或事件。为此，请选择相应的按钮，然后选择警报状态和要执行的操作。

如果您选择 Lambda 函数作为警报操作，则需要指定函数名称或 ARN，并且可以选择该函数的特定版本。

有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

> **Note**
>
> 要创建执行 AWS Systems Manager Incident Manager 操作的警报，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。

12. 选择下一步。
13. 在 Name and description（名称和描述）下，输入告警的名称和描述，然后选择 Next（下一步）。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。

> **Tip**
>
> 告警名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符

14. 在 Preview and create（预览和创建）下，确认告警的信息和条件正确，然后选择 Create alarm（创建告警）。

修改异常检测模型

创建告警后，可以调整异常检测模型。您可以排除某些时间段，不在创建模型时使用。从训练数据中排除系统中断、部署和假日等异常事件至关重要。您还可以指定是否针对夏令时更改调整模型。

针对告警调整异常检测模型

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。
3. 选择警报的名称。如有必要，请使用搜索框来查找警报。
4. 依次选择分析、在指标中。
5. 在详细信息列中，依次选择 ANOMALY_DETECTION_BAND、编辑异常检测模型。
6. 要排除用于生成模型的时间段，请按结束日期选择日历图标。然后，选择或输入要从训练中排除的天数和时间，并选择 Apply（应用）。

7. 如果指标需要区分夏令时更改，请在 Metric timezone（指标时区）框中选择相应的时区。
8. 选择 Update（更新）。

删除异常检测模型

对告警使用异常检测会产生 费用。作为最佳实践，如果警报不再需要异常检测模型，请先删除警报，然后删除模型。评估异常检测警报时，系统将代表您创建任何缺失的异常检测器。如果您删除模型而不删除告警，则告警会自动重新创建模型。

删除警报

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。
3. 选择警报的名称。
4. 依次选择 Actions（操作）和 Delete（删除）。
5. 在确认框中，选择删除。

要删除曾用于告警的异常检测模型

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标），然后选择 All metrics（所有指标）。
3. 选择 Browse（浏览），然后选择包含异常检测模型的指标。您可以在搜索框中搜索您的指标，也可以通过选择选项来选择您的指标。
   - （可选）如果要使用原始界面，请选择 All metrics（所有指标），然后选择包含异常检测模型的指标。您可以在搜索框中搜索您的指标，也可以通过选择选项来选择您的指标。
4. 选择 Graphed metrics（绘制的指标）。
5. 在 Graphed metrics（绘成图表的指标）选项卡中，选择要删除的异常检测模型的名称，然后选择 Delete anomaly detection model（删除异常检测模型）。
   - （可选）如果要使用原始界面，请选择 Edit model（编辑模型）。系统会将您定向到新屏幕。在新屏幕上，选择 Delete model（删除模型），然后选择 Delete（删除）。

根据日志触发警报

以下各节中的步骤说明了如何创建监测日志的 CloudWatch 警报。

根据日志组指标筛选条件创建 CloudWatch 告警

本节中的过程介绍如何根据日志组指标筛选器创建告警。使用指标筛选条件，您可以在日志数据发送到 CloudWatch 时查找其中的字词和模式。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 使用筛选条件从日志事件创建指标。在根据日志组指标筛选器创建告警之前，您必须完成以下操作：

- 创建 日志组。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 使用日志组和日志流。
- 创建指标筛选条件。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 为日志组创建指标筛选条件。

要根据日志组指标筛选条件创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Logs（日志），然后选择 Log groups（日志组）。
3. 选择包含您的指标筛选器的日志组。
4. 选择 Metric filters（指标筛选条件）。
5. 在指标筛选器选项卡中，选中要用于创建告警的指标筛选器对应的复选框。
6. 选择创建警报。
7. （可选）在 Metric（指标）下，编辑 Metric name（指标名称）、Statistic（统计数据）和 Period（周期）。
8. 在条件下面，指定以下内容：
   a. 对于 Threshold type（阈值类型），选择 Static（静态）或 Anomaly detection（异常检测）。
   b. 对于 Whenever ***your-metric-name*** is . . .（每当 your-metric-name . . .），选择 Greater（大于）、Greater/Equal（大于/等于）、Lower/Equal（小于/等于）或 Lower（小于）。
   c. 对于 than . . .（相比 . . .），为您的阈值指定一个数值。
9. 选择其他配置。
   a. 对于 Data points to alarm（触发告警的数据点数），指定多少数据点会触发您的告警进入 ALARM 状态。如果指定匹配的值，则如果多个连续评估期违例，该告警将变为 ALARM 状态。

要创建 M-out-of-N（M（最大为 N））告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅使用 Amazon CloudWatch 告警。

b. 对于 Missing data treatment（缺失数据处理），选择一个选项以指定在评估告警时如何应对缺失数据。

10. 选择下一步。
11. 对于 Notification（通知），选择当您的告警处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时要通知的 Amazon SNS 主题。

    a. （可选）要为相同告警状态或不同告警状态发送多个通知，请选择 Add notification（添加通知）。

    b. （可选）要想不发送通知，请选择 Remove（删除）。

12. 要让警报执行 Auto Scaling、EC2、Lambda 或 Systems Manager 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。如果您选择 Lambda 函数作为警报操作，则需要指定函数名称或 ARN，并且可以选择该函数的特定版本。

    告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。

13. 选择下一步。
14. 对于名称和描述，输入告警的名称和描述。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。
15. 对于 Preview and create（预览并创建），确保您的配置正确，然后选择 Create alarm（创建警告）。

组合警报

借助 CloudWatch，您可以将多个警报组合成一个复合警报，从而针对整个应用程序或一组资源创建汇总的运行状况聚合指标。复合警报可通过监控其他警报的状态来确定其状态。您可以定义规则，以使用布尔逻辑来组合这些受监控警报的状态。

您可以仅在聚合级别执行操作，使用复合警报来减少警报噪音。例如，您可以创建一个复合警报，在触发任何与 Web 服务器相关的警报时向 Web 服务器团队发送通知。当其中任何一个警报进入“ALARM”状态时，复合警报会自行进入“ALARM”状态，并向您的团队发送通知。如果与您的 Web 服务器相关的其他警报也进入“ALARM”状态，则您的团队不会收到过多的新通知，因为复合警报已经将当前情况告知团队成员。

您还可以使用复合警报来创建复杂的警报条件，并仅在满足许多不同条件时才执行操作。例如，您可以创建一个复合警报，将 CPU 警报和内存警报结合在一起，并且只有在 CPU 和内存警报都触发时才通知您的团队。

使用复合警报

使用复合警报时，您有两个选项：

- 配置您只想在复合警报级别执行的操作，并在不执行任何操作的情况下创建底层受监控警报
- 在复合警报级别配置一组不同的操作。例如，如果问题普遍存在，复合警报操作可能会让其他团队参与进来。

复合警报仅可执行以下操作：

- 通知 Amazon SNS 主题
- 调用 Lambda 函数
- 在 Systems Manager Ops Center 中创建 OpsItems
- 在 Systems Manager Incident Manager 中创建事件

> **Note**
>
> 复合警报中的所有基础警报都必须与复合警报位于相同的账户和区域中。但是，如果您在 CloudWatch 跨账户可观测性监控账户中设置复合告警，则基础告警可以监视不同源账户和该监控账户本身的指标。有关更多信息，请参阅 CloudWatch 跨账户可观测性。
> 单个复合告警可以监控 100 个基础告警，150 个复合告警可以监控单个基础告警。

规则表达式

所有复合告警都包含规则表达式。规则表达式告诉复合告警要监控哪些其他告警并确定其状态。规则表达式可以同时引用指标告警和复合告警。当您在规则表达式中引用告警时，您可以为告警指定一个函数，用于确定告警将处于以下三种状态中的哪一种：

- 告警

  如果告警处于 ALARM 状态，则 ALARM ("alarm-name or alarm-ARN") 为 TRUE。

- OK

  如果告警处于 OK 状态，则 OK ("alarm-name or alarm-ARN") 为 TRUE。

- INSUFFICIENT_DATA

  如果指定的告警处于 INSUFFICIENT_DATA 状态，则 INSUFFICIENT_DATA ("alarm-name or alarm-ARN") 为 TRUE。

> **Note**
>
> TRUE 始终计算为 TRUE，FALSE 始终计算为 FALSE。

表达式示例

请求参数 `AlarmRule` 支持使用逻辑运算符 `AND`、`OR` 和 `NOT`，这样您就可以将多个函数组合成一个表达式。以下示例表达式显示了如何在复合告警中配置基础告警：

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)`

  该表达式指定复合告警仅在 `CPUUtilizationTooHigh` 和 `DiskReadOpsTooHigh` 处于 `ALARM` 状态时进入 `ALARM` 状态。

- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)`

  该表达式指定复合告警在 `CPUUtilizationTooHigh` 处于 `ALARM` 且 `DeploymentInProgress` 处于 `ALARM` 状态时进入 `ALARM` 状态。这是一个复合告警的示例，它降低了部署时段内的告警噪音。

- `(ALARM(CPUUtilizationTooHigh) OR ALARM(DiskReadOpsTooHigh)) AND OK(NetworkOutTooHigh)`

  该表达式指定复合告警在 `(ALARM(CPUUtilizationTooHigh)` 或 `(DiskReadOpsTooHigh)` 处于 `ALARM` 且 `(NetworkOutTooHigh)` 处于 `OK` 状态时进入 `ALARM` 状态。这是一个复合告警的示例，如果任何一个基础告警在发生网络问题时不处于 `ALARM` 状态，则它不会向您发送通知，从而降低告警噪音。

主题

- 创建复合告警
- 抑制复合警报操作

创建复合告警

本部分中的步骤说明了如何使用 CloudWatch 控制台创建复合告警。您还可以使用 API 或 AWS CLI 创建复合告警。有关更多信息，请参阅 PutCompositeAlarm 或 put-composite-alarm

创建复合告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Alarms（告警），然后选择 All alarms（所有告警）。
3. 在告警列表中，选中要在规则表达式中引用的每个现有告警旁边的复选框，然后选择 Create composite alarm（创建复合告警）。
4. 在 Specify composite alarm conditions（指定复合告警条件）中，指定新复合告警的规则表达式。

> **Note**
>
> 您从告警列表中选择的告警将自动列在 Conditions（条件）框中。默认情况下，ALARM 函数已指定给您的每个告警，并且每个告警都由逻辑运算符 OR 连接。

您可以使用以下子步骤修改规则表达式：

a. 您可以将每个告警的所需状态从 ALARM 更改为 OK 或 INSUFFICENT_DATA。
b. 您可以将规则表达式中的逻辑运算符从 OR 更改为 AND 或 NOT，并且您可以添加括号来对函数进行分组。
c. 您可以在规则表达式中包含其他告警，也可以从规则表达式中删除告警。

Example: Rule expression with conditions（示例：带条件的规则表达式）

```
(ALARM("CPUUtilizationTooHigh") OR
ALARM("DiskReadOpsTooHigh")) AND
OK("NetworkOutTooHigh")
```

复合告警在 ALARM（“CPUUtilizationTooHigh”）或 ALARM（“DiskReadOpsTooHigh”）处于 ALARM 状态且同时 OK（“NetworkOutTooHigh”）处于 OK 状态时进入 ALARM 的情况下的示例规则表达式。

5. 在完成后，选择下一步。
6. 在 Configure actions（配置操作）下，您可以从以下选项中进行选择：

   对于Notification（通知）

   - Select an exisiting SNS topic（选择一个现有的 SNS 主题）、Create a new SNS topic（创建新 SNS 主题），或者 Use a topic ARN（使用主题 ARN）定义将接收通知的 SNS 主题。
   - Add notification（添加通知），以便为相同告警状态或不同告警状态发送多个通知。
   - Remove（删除）以阻止您的告警发送通知或采取措施。

   （可选）要让警报在状态更改时调用 Lambda 函数，请选择添加 Lambda 操作。然后指定函数名称或 ARN，也可以选择该函数的特定版本。

   对于 Systems Manager action（Systems Manager 操作）

   - Add Systems Manager action（添加 Systems Manager 操作），以便您的告警可以在进入 ALARM 状态时执行 SSM 操作。

   要了解有关 Systems Manager 操作的更多信息，请参阅《AWS Systems Manager 用户指南》中的 配置 CloudWatch 以从告警中创建 OpsItems 和《Incident Manager 用户指南》中的 事件创建。要创建执行 SSM Incident Manager 操作的告警，您必须具有正确的权限。有关更多信息，请参阅《Incident Manager 用户指南》中的 AWS Systems Manager Incident Manager 的基于身份的策略示例。
7. 在完成后，选择下一步。
8. 在 Add name and description（添加名称和描述）下，为您的新复合告警输入告警名称和可选描述。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。
9. 在完成后，选择下一步。
10. 在 Preview and create（预览和创建）下，确认您的信息，然后选择 Create composite alarm（创建复合告警）。

> **Note**
>
> 您可以创建一个复合告警循环，其中一个复合告警和另一个复合告警相互依赖。如果处于这种情况，则复合告警将停止被评估，并且您无法删除复合告警，因为它们相互依赖。要打破复合告警之间的依赖循环，最简单的方法是将您的其中一个复合告警中的函数 `AlarmRule` 更改为 `False`。

抑制复合警报操作

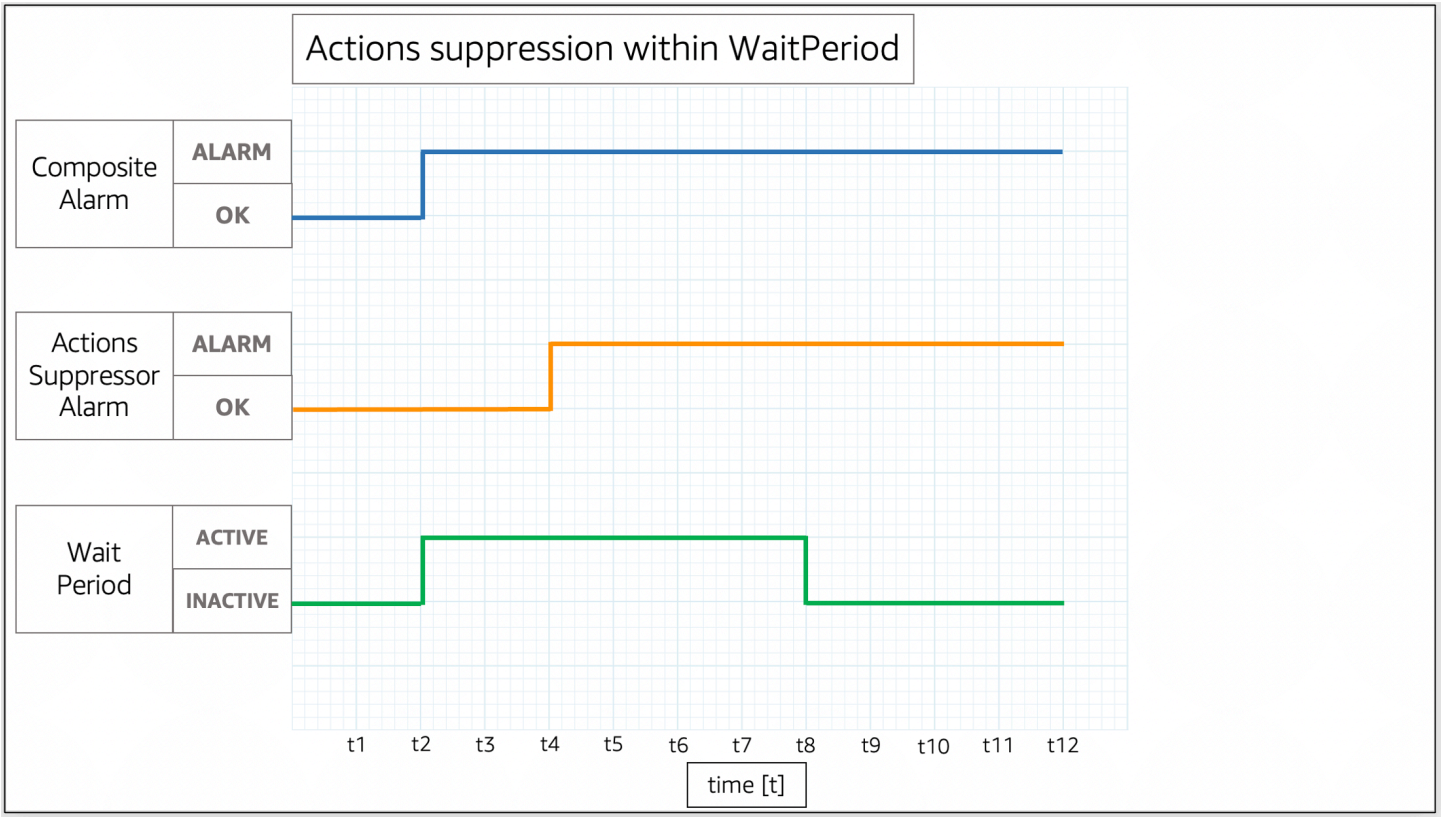
复合警报允许您在多个警报中获得运行状况的聚合视图，因此存在一些预计会触发这些警报的常见情况。例如，在应用程序的维护时段或调查正在发生的事件时。在这种情况下，您可能需要抑制复合警报的操作，以防止不必要的通知或创建新的事件工单

通过复合告警操作抑制，您可以将告警定义为抑制器告警。抑制器告警可防止复合告警采取行动。例如，您可以指定表示支持资源状态的抑制器告警。如果支持资源关闭，则抑制器告警会阻止复合告警发送通知。复合告警操作抑制功能可帮助您降低警报噪音，从而减少管理告警的时间，将更多的时间集中在操作上。

您可以在配置复合告警时指定抑制器告警。任何告警都可以用作抑制器告警。当抑制器告警的状态从 OK 变为 ALARM 时，其复合告警停止执行操作。当抑制器告警的状态从 ALARM 变为 OK 时，其复合告警恢复执行操作。

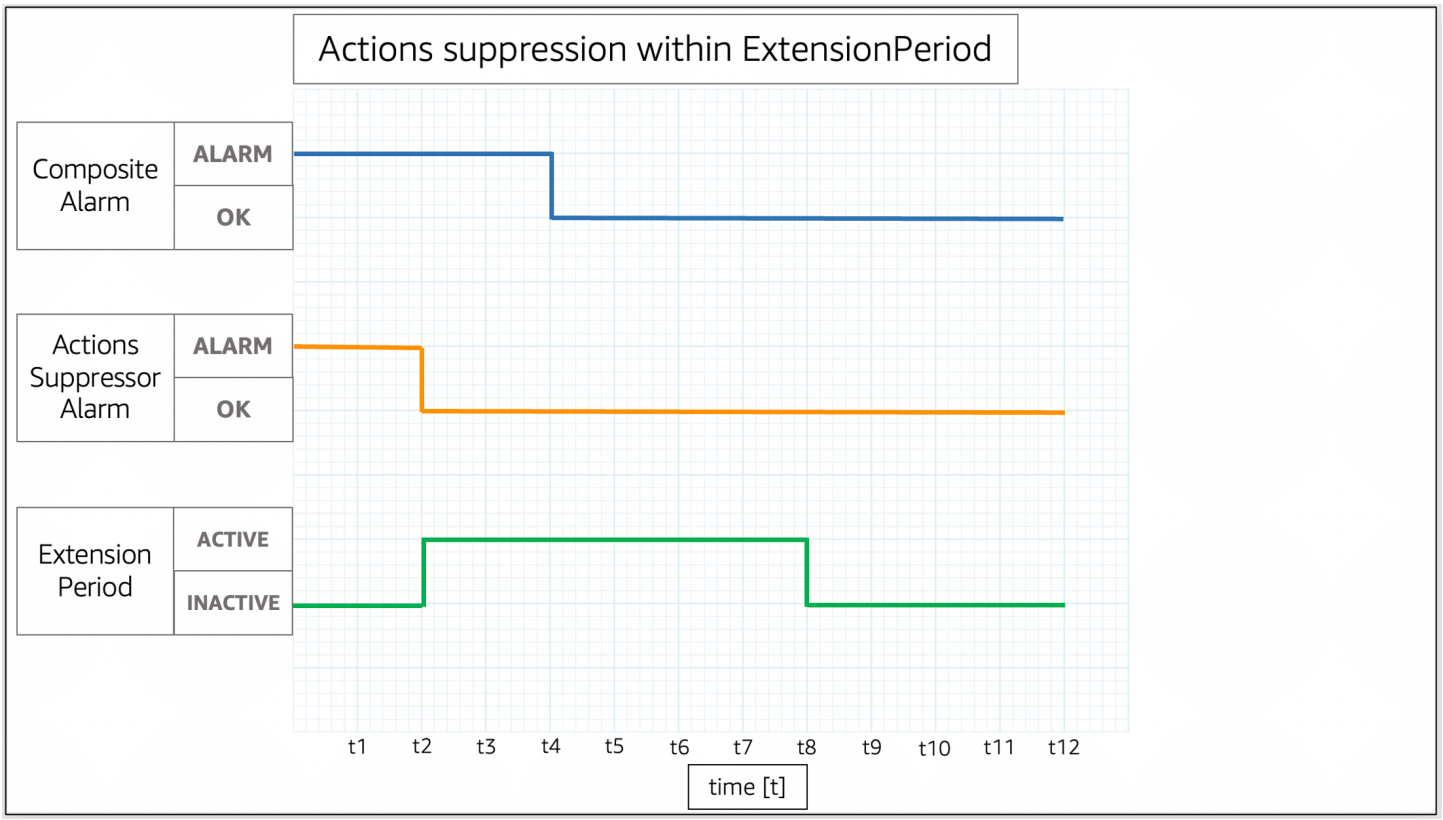
`WaitPeriod` 和 `ExtensionPeriod`

指定抑制器告警时，需要设置参数 `WaitPeriod` 和 `ExtensionPeriod`。这些参数可防止复合告警在抑制器告警改变状态时意外执行操作。使用 `WaitPeriod` 补偿当抑制器告警从 OK 变为 ALARM 时发生的任何延迟。例如，如果抑制器告警在 60 秒内从 OK 变为 ALARM，则将 `WaitPeriod` 设置为 60 秒。



在图像中，复合告警在 t2 时从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t8 时结束。这使抑制器告警有时间在 t4 时从状态 OK 变为 ALARM，然后在 WaitPeriod 在 t8 时到期时抑制复合告警的操作。

使用 ExtensionPeriod 补偿当抑制器告警变为 OK 后复合告警变为 OK 时发生的任何延迟。例如，如果复合告警在抑制器告警变为 OK 的 60 秒内变为 OK，则将 ExtensionPeriod 设置为 60 秒。



在图像中，抑制器告警在 t2 时从 ALARM 变为 OK。ExtensionPeriod 在 t2 时开始并在 t8 时结束。这使得复合告警有时间从 ALARM 变为 OK，然后 ExtensionPeriod 在 t8 时到期。

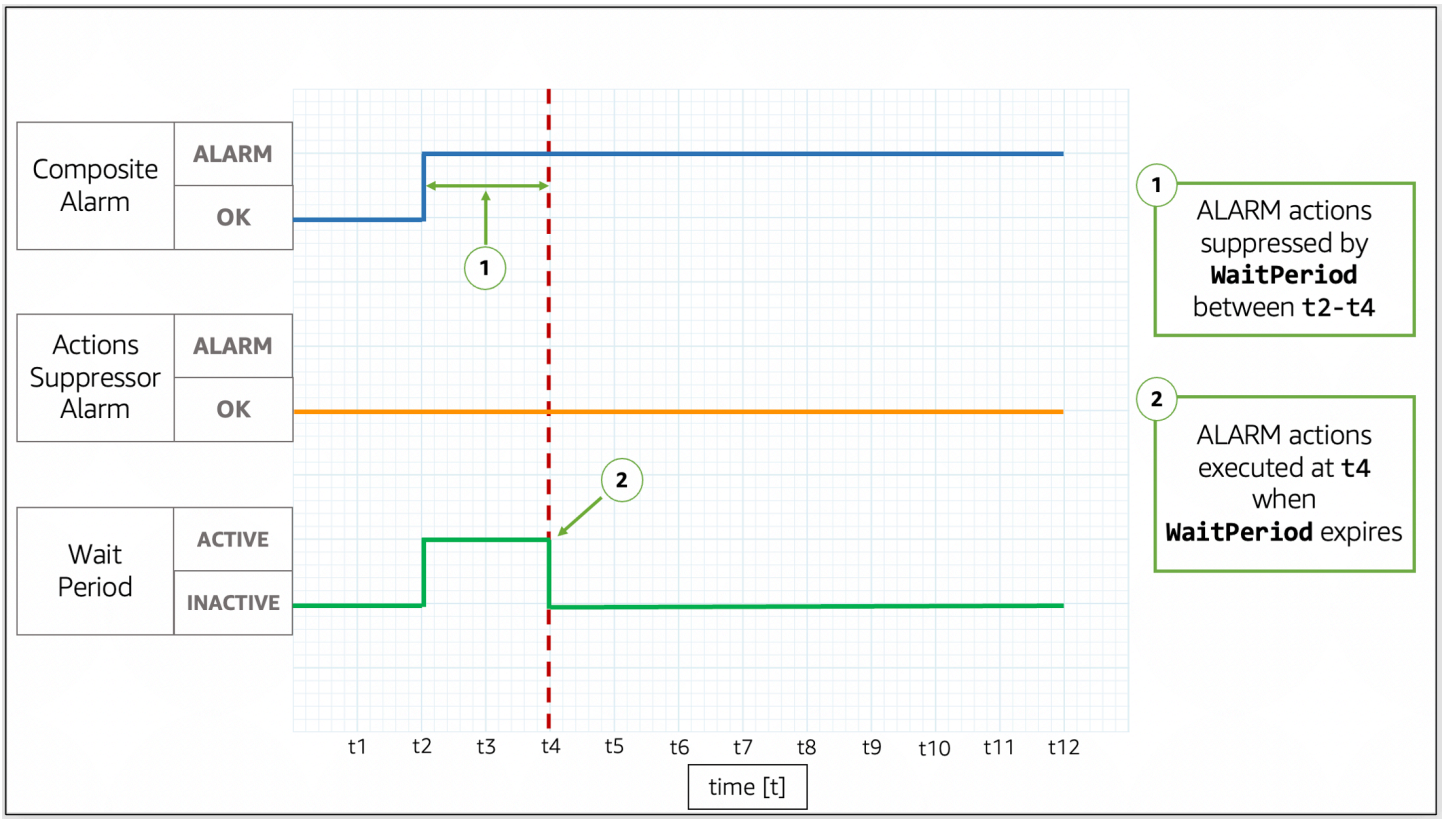
复合告警在 WaitPeriod 和 ExtensionPeriod 变为活动状态时不执行操作。当 ExtensionPeriod 和 WaitPeriod 变为非活动状态时，复合告警基于其当前状态执行操作。我们建议您将每个参数的值设置为 60 秒，因为 CloudWatch 每分钟评估一次指标告警。您可以将参数设置为任意整数（以秒为单位）。

以下示例更详细地描述了 WaitPeriod 和 ExtensionPeriod 如何防止复合告警意外执行操作。

> **Note**
>
> 在以下示例中，WaitPeriod 配置为 2 个时间单位，ExtensionPeriod 配置为 3 个时间单位。

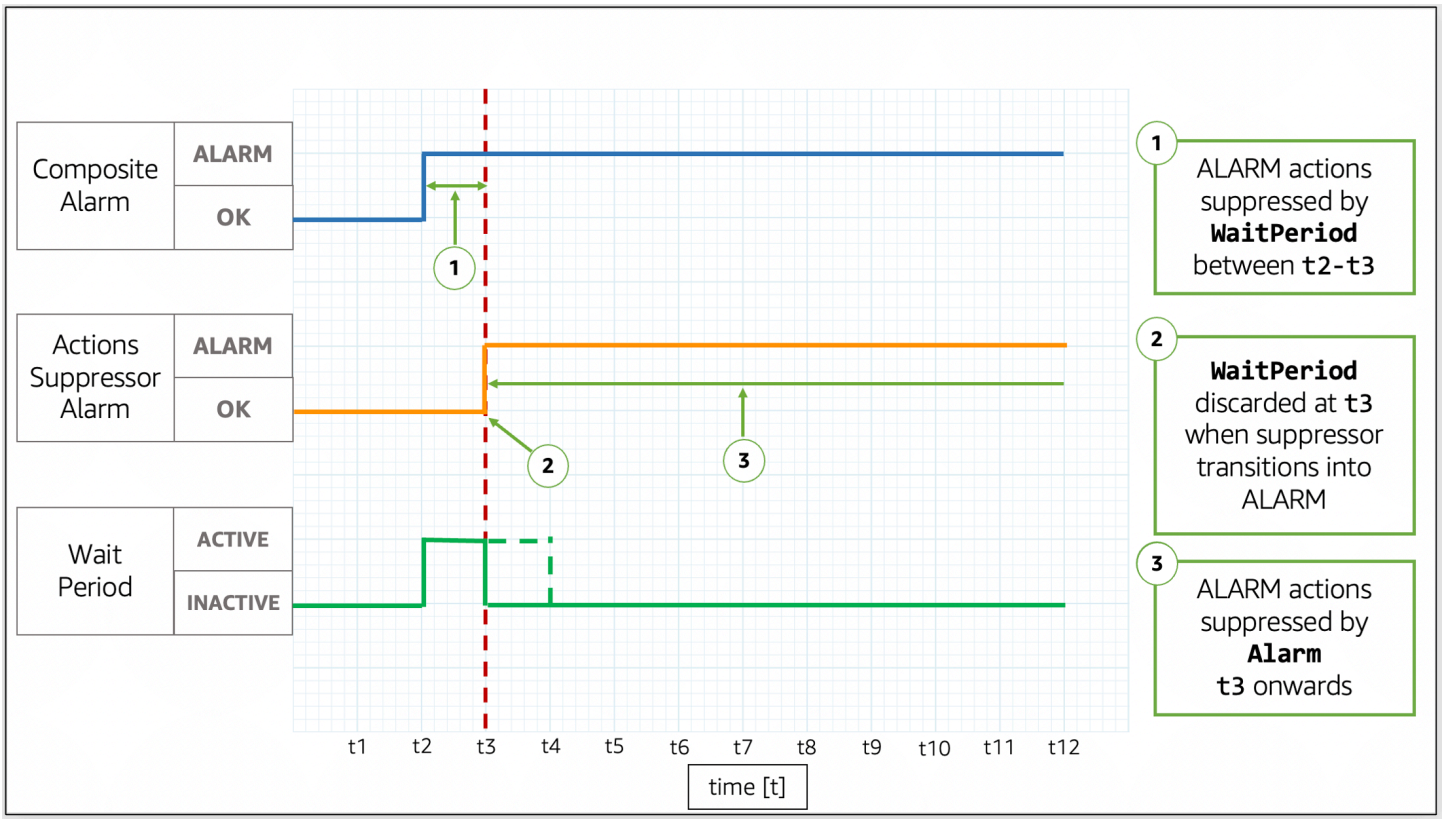
示例

示例 1：操作在 **WaitPeriod** 后未被抑制



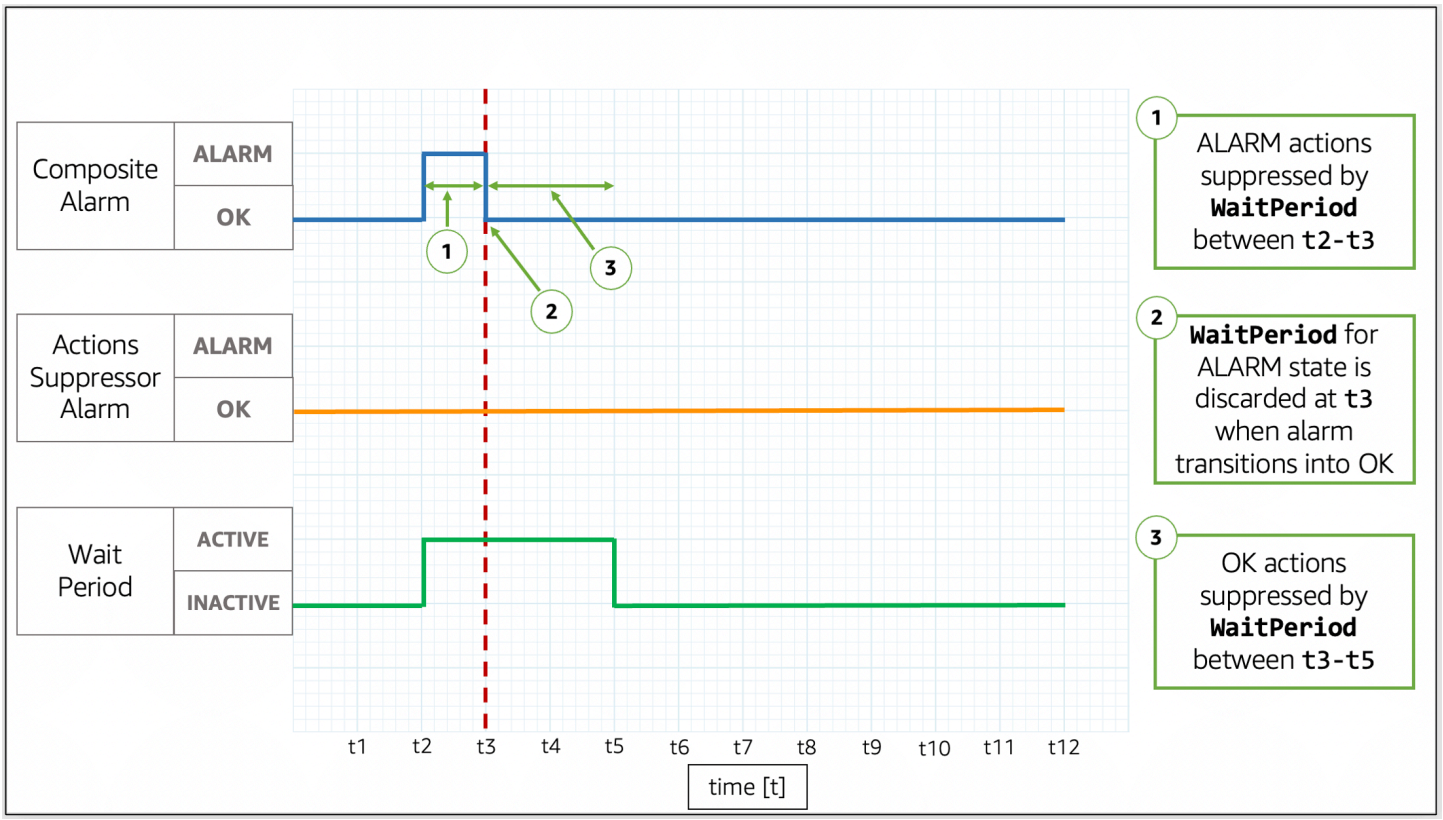
在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t4 时结束，因此它可以阻止复合告警执行操作。WaitPeriod 在 t4 时到期后，复合告警会执行操作，因为抑制器告警仍处于 OK 状态。

示例 2：操作在 **WaitPeriod** 到期前被告警抑制



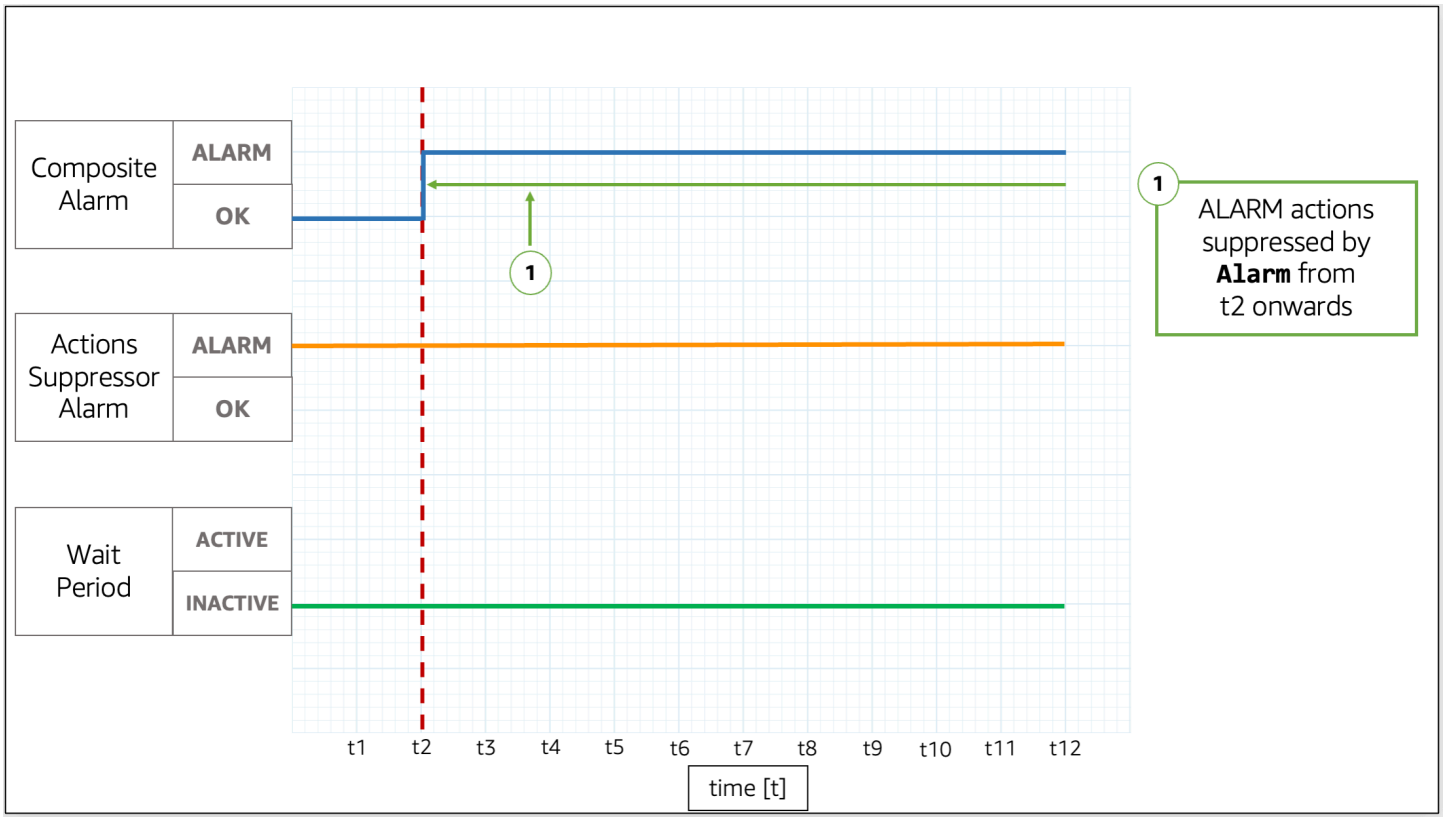
在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t4 时结束。这使抑制器告警有时间在 t3 时从状态 OK 变为 ALARM。因为抑制器告警的状态在 t3 时从 OK 变为 ALARM，从 t2 开始的 WaitPeriod 将被丢弃，抑制器告警现在会阻止复合告警执行操作。

示例 3：操作被 **WaitPeriod** 抑制时的状态转换



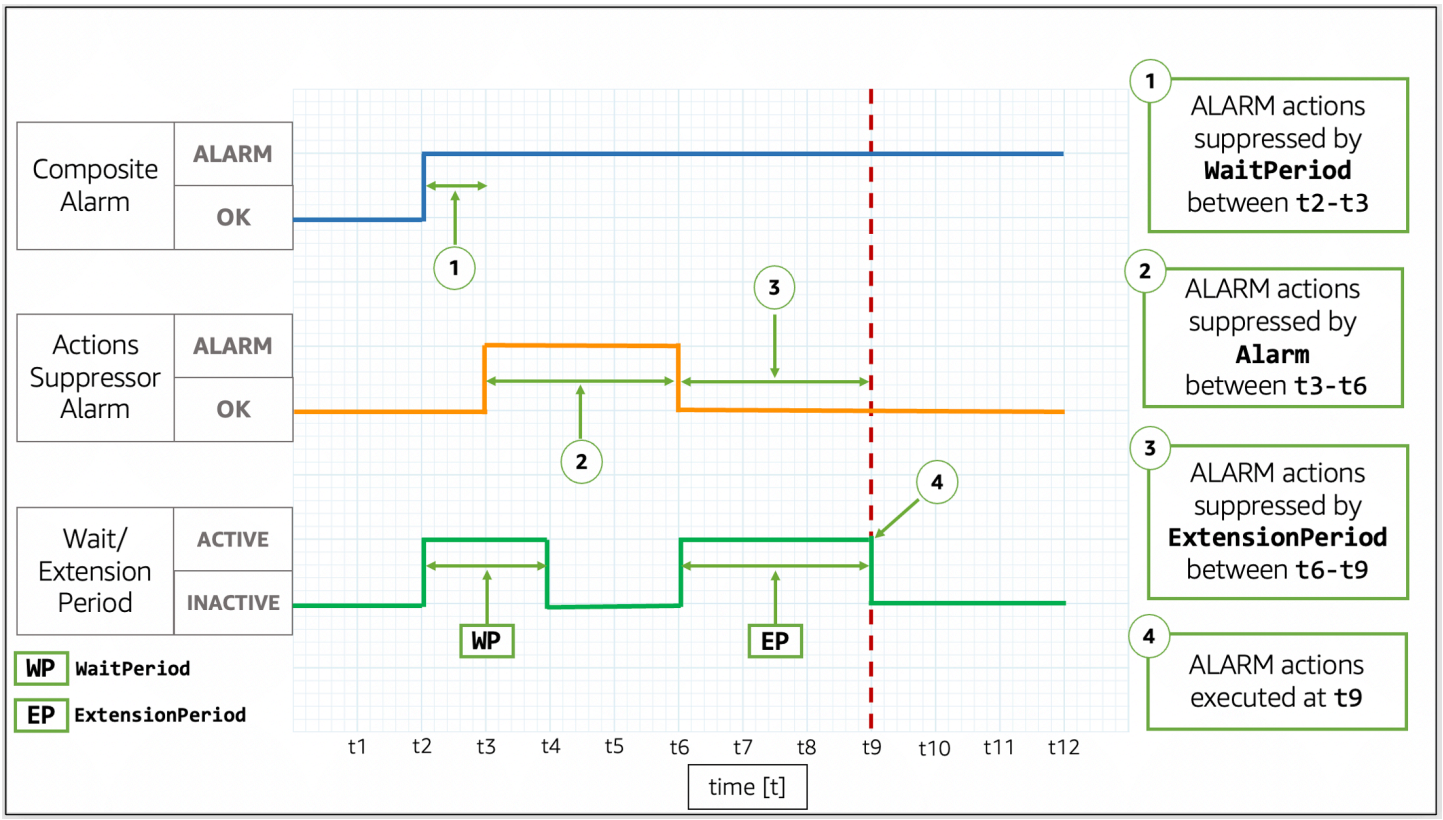
在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t4 时结束。这使抑制器告警有时间改变状态。复合告警在 t3 时变回 OK，所以在 t2 时开始的 WaitPeriod 被丢弃。新的 WaitPeriod 在 t3 时开始并在 t5 时结束。新的 WaitPeriod 在 t5 时到期后，复合告警将执行操作。

Example 4: State transition when actions are suppressed by alarm（示例 4：操作被告警抑制时的状态转换）



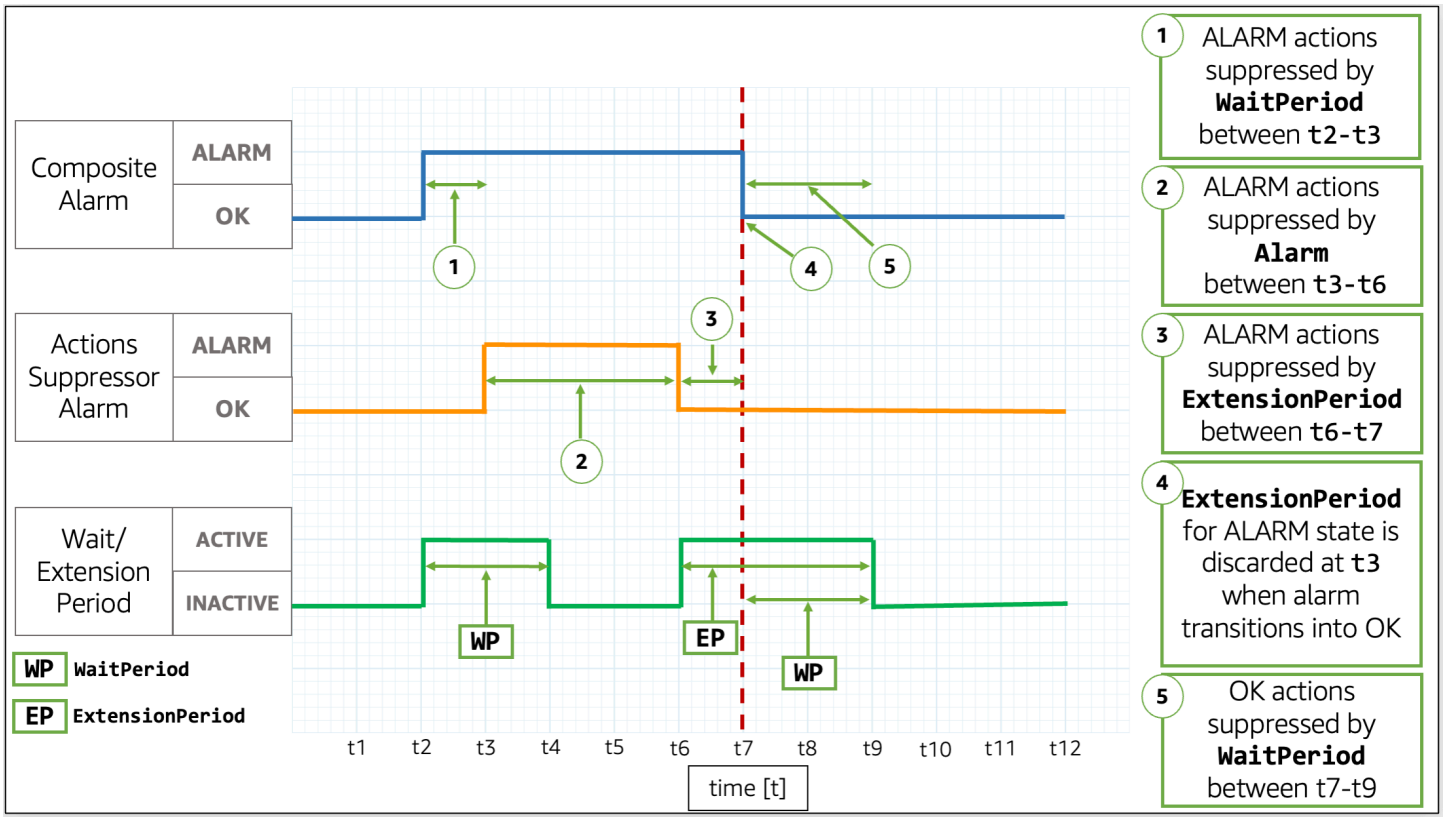
在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。抑制器告警已经处于 ALARM 状态。抑制器告警可防止复合告警执行操作。

示例 5：操作在 **`ExtensionPeriod`** 后未被抑制



在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t4 时结束。这使抑制器告警有时间在 t3 时从状态 OK 变为 ALARM，然后在 t6 前抑制复合告警的操作。因为抑制器告警的状态在 t3 时从 OK 变为 ALARM，从 t2 开始的 WaitPeriod 将被丢弃。在 t6 时，抑制器告警变为 OK。ExtensionPeriod 在 t6 时开始并在 t9 时结束。ExtensionPeriod 到期后，复合告警将执行操作。

示例 6：操作被 **ExtensionPeriod** 抑制时的状态转换



在图像中，复合告警在 t2 时状态从 OK 变为 ALARM。WaitPeriod 在 t2 时开始并在 t4 时结束。这使抑制器告警有时间在 t3 时从状态 OK 变为 ALARM，然后在 t6 前抑制复合告警的操作。因为抑制器告警的状态在 t3 时从 OK 变为 ALARM，从 t2 开始的 WaitPeriod 将被丢弃。在 t6 时，抑制器告警变回 OK。ExtensionPeriod 在 t6 时开始并在 t9 时结束。当复合告警在 t7 时变回 OK 时，ExtensionPeriod 被丢弃，并且新的 WaitPeriod 在 t7 时开始并在 t9 时结束。

> **Tip**
>
> 如果您更换操作抑制器告警，则任何活动的 WaitPeriod 或 ExtensionPeriod 将被丢弃。

根据警报更改执行操作

CloudWatch 可以在发生两种类型的警报更改时通知用户：警报状态更改和警报配置更新。

警报在进行评估时，其状态可能会变为其他状态，例如“警报”、“正常”或“数据不足”。这些警报状态更改可能表示可能发生了事故、已恢复正常或指标不可用。在这种情况下，您可能需要使用下面的任何一种方法来吸引或通知用户：

- 您可以将警报配置为向 SNS 主题发送通知，以作为警报操作的内容。然后可以为应用程序对应用程序（A2A）的消息收发和应用程序对个人（A2P）的通知配置 SNS 主题，包括电子邮件通知和短信等渠道。您为 SNS 主题定义的所有目标都会收到警报通知。有关更多信息，请参阅 Amazon SNS 事件目标。
- 您可以为警报状态更改事件配置通知。AWS配置此类通知的最直接方式是使用 User Notifications，这也是推荐的方法。

此外，每当警报状态更改以及创建、删除或更新警报时，CloudWatch 都会将事件发送到 Amazon EventBridge。您可以编写 EventBridge 规则，从而在 EventBridge 收到这些事件时执行操作或通知您。

主题

- 将警报更改通知用户
- 告警事件和 EventBridge

将警报更改通知用户

本节说明了如何使用 AWS User Notifications 或 Amazon Simple Notification Service，来让用户收到有关警报更改的通知。

设置 AWS User Notifications

您可以使用 AWS User Notifications 来设置传输渠道，以用于接收有关 CloudWatch 警报状态更改和配置更改事件的通知。当事件与指定的规则匹配时，会收到通知。您可以通过多个渠道接收事件通知，包括电子邮件、AWS Chatbot 聊天通知或 AWS 控制台移动应用程序推送通知。您还可以在 控制台通知中心 中查看通知。用户通知支持聚合，这可以减少在具体事件期间收到的通知数量。

您使用 AWS User Notifications 创建的通知配置不会计入您可以为每个目标警报状态配置的操作数量限制。当 AWS User Notifications 对发送给 Amazon EventBridge 的事件进行匹配时，除非您指定高级筛选条件来将特定的警报或模式列入允许列表或拒绝列表，否则会针对您账户和选定区域中的所有警报发送通知。

以下高级筛选条件示例将对名为 `ServerCpuTooHigh` 的警报从“正常”变为“警报”的警报状态更改进行匹配。

```
{
"detail": {
    "alarmName": ["ServerCpuTooHigh"],
```

```
    "previousState": { "value": ["OK"] },
    "state": { "value": ["ALARM"] }
  }
}
```

您可以使用警报在 EventBridge 事件中发布的任何属性来创建筛选条件。有关更多信息，请参阅 告警事件和 EventBridge。

设置 Amazon SNS 通知

您可以使用 Amazon Simple Notification Service 来发送应用程序到应用程序（A2A）消息和应用程序对人（A2P）消息，包括手机短信（SMS）和电子邮件。有关更多信息，请参阅 Amazon SNS 事件目标。

对于警报可能处于的每种状态，您都可以将警报配置为向某个 SNS 主题发送消息。您为给定警报状态配置的每个 Amazon SNS 主题，都将计入您可以为该警报和状态配置的操作数量限制。您可以从账户中的任何警报向同一 Amazon SNS 主题发送消息，并为应用程序（A2A）和个人（A2P）消费端使用相同的 Amazon SNS 主题。由于此配置是在警报级别进行的，因此只有您配置的警报才会向所选的 Amazon SNS 主题发送消息。

首先，创建一个主题，然后订阅此主题。您可以选择将测试消息发布到此主题。有关示例，请参阅使用 AWS Management Console 设置 Amazon SNS 主题。有关更多信息，请参阅 Amazon SNS 入门。

或者，如果您计划使用 AWS Management Console来创建 CloudWatch 警报，则可跳过此过程，因为您可在创建警报时创建主题。

创建 CloudWatch 警报时，您可以为任何警报目标状态添加操作。为您想要收到通知的状态添加 Amazon SNS 通知，然后选择您在上一步中创建的 Amazon SNS 主题，从而在警报进入选定状态时发送电子邮件通知。

> **Note**
>
> 在创建 Amazon SNS 主题时，您可以选择将其设为标准主题或 FIFO 主题。CloudWatch 定会将所有告警通知发布到这两种类型的主题。但是，即使您使用 FIFO 主题，在极少数情况下，CloudWatch 也会不按顺序将通知发送到该主题。如果您使用 FIFO 主题，告警会将告警通知的消息组 ID 设置为告警的 ARN 的哈希值。

主题

- 防止混淆代理安全问题

- 使用 AWS Management Console 设置 Amazon SNS 主题
- 使用 AWS CLI 设置 SNS 主题

防止混淆代理安全问题

混淆代理问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆代理问题。一个服务（呼叫服务）调用另一项服务（所谓的服务）时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

建议在资源策略中使用 aws:SourceArn、aws:SourceAccount、aws:SourceOrgID 和 aws:SourceOrgPaths 全局条件上下文键，以限制 Amazon SNS 为其他服务提供的资源访问权限。使用 aws:SourceArn 来仅将一个资源与跨服务访问相关联。使用 aws:SourceAccount 来让该账户中的任何资源与跨服务使用相关联。使用 aws:SourceOrgID 来允许某组织内的任何账户中的任何资源与跨服务使用相关联。使用 aws:SourceOrgPaths 来将 AWS Organizations 路径内账户中的任何资源与跨服务使用相关联。有关使用和了解路径的更多信息，请参阅了解 AWS Organizations 实体路径。

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符 (*) 的 aws:SourceArn 全局上下文条件键。例如，arn:aws:*servicename*:*:*123456789012*:*。

如果 aws:SourceArn 值不包含账户 ID，例如 Amazon S3 桶 ARN，您必须使用 aws:SourceAccount 和 aws:SourceArn 来限制权限。

要防范大规模混淆代理问题，请在基于资源的策略中将 aws:SourceOrgID 或 aws:SourceOrgPaths 全局条件上下文键与资源的组织 ID 或组织路径一起使用。包含 aws:SourceOrgID 或 aws:SourceOrgPaths 键的策略将自动包含正确的账户，并且当您在组织中添加、删除或移动账户时，无需手动更新策略。

aws:SourceArn 的值必须是发送通知的警报的 ARN。

以下示例演示如何在 CloudWatch 中使用 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键来防范混淆代理问题。

```
{
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
```

```
            "Service": "cloudwatch.amazonaws.com"
        },
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
            },
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            }
        }
    }]
}
```

如果警报 ARN 包含任何非 ASCII 字符，请仅使用 `aws:SourceAccount` 全局条件键限制权限。

使用 AWS Management Console 设置 Amazon SNS 主题

首先，创建一个主题，然后订阅此主题。您可以选择将测试消息发布到此主题。

创建 SNS 主题

1. 通过 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在 Amazon SNS 控制面板上的 Common actions（常用操作）下，选择 Create Topic（创建主题）。
3. 在 Create new topic（创建新主题）对话框中，为 Topic name（主题名称）输入主题的名称（例如 **`my-topic`**）。
4. 选择 Create topic（创建主题）。
5. 复制下一个任务的 Topic ARN（主题 ARN）（例如 arn:aws:sns:us-east-1:111122223333:my-topic）。

订阅 SNS 主题

1. 通过 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在导航窗格中，依次选择 Subscriptions（订阅）、Create subscription（创建订阅）。
3. 在 Create subscription（创建订阅）对话框中，为 Topic ARN（主题 ARN）粘贴在上一任务中创建的主题 ARN。
4. 对于协议，选择电子邮件。

5. 对于 Endpoint（端点），输入一个可用于接收通知的电子邮件地址，然后选择 Create subscription（创建订阅）。
6. 从您的电子邮件应用程序中，打开来自 AWS 通知的消息并确认您的订阅。

   您的 Web 浏览器将显示来自 Amazon SNS 的确认响应。

向 SNS 主题发布测试消息

1. 通过 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在导航窗格中，选择 Topics（主题）。
3. 在 Topics（主题）页面上，选择一个主题，然后选择 Publish to topic（发布到主题）。
4. 在 Publish（发布消息）页面中，为 Subject（主题）输入消息的主题行，并为 Message（消息）输入简短的消息。
5. 选择 Publish Message（发布消息）。
6. 查看电子邮件，确认您已收到消息。

使用 AWS CLI 设置 SNS 主题

首先，您创建一个 SNS 主题，然后将一条消息直接发布到该主题，以测试您是否正确配置了该主题。

设置 SNS 主题

1. 使用 create-topic 命令创建主题，如下所示。

   ```
   aws sns create-topic --name my-topic
   ```

   Amazon SNS 返回具有以下格式的主题 ARN：

   ```
   {
       "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"
   }
   ```

2. 使用 subscribe 命令以通过您的电子邮件地址订阅该主题。如果订阅请求成功，您将收到一封确认电子邮件。

   ```
   aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --
   protocol email --notification-endpoint my-email-address
   ```

Amazon SNS 将返回以下内容：

```
{
    "SubscriptionArn": "pending confirmation"
}
```

3. 从您的电子邮件应用程序中，打开来自 AWS 通知的消息并确认您的订阅。

   您的 Web 浏览器将显示来自 Amazon Simple Notification Service 的确认响应。

4. 使用 list-subscriptions-by-topic 命令检查订阅。

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS 将返回以下内容：

```
{
  "Subscriptions": [
    {
        "Owner": "111122223333",
        "Endpoint": "me@mycompany.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
        "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
  ]
}
```

5. （可选）使用 publish 命令向主题发布测试消息。

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS 将返回以下内容。

```
{
    "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. 查看电子邮件，确认您已收到消息。

警报状态更改时的 Amazon SNS 通知架构

本节列举了警报状态更改时向 Amazon SNS 主题发送的通知的架构。

指标警报状态更改时的架构

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "AlarmConfigurationUpdatedTimestamp": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OldStateValue": "string",
  "OKActions": ["string"],
  "AlarmActions": ["string"],
  "InsufficientDataActions": ["string"],
  "Trigger": {
    "MetricName": "string",
    "Namespace": "string",
    "StatisticType": "string",
    "Statistic": "string",
    "Unit": "string or null",
    "Dimensions": [
      {
        "value": "string",
        "name": "string"
      }
    ],
    "Period": "integer",
    "EvaluationPeriods": "integer",
    "DatapointsToAlarm": "integer",
    "ComparisonOperator": "string",
    "Threshold": "number",
    "TreatMissingData": "string",
    "EvaluateLowSampleCountPercentile": "string or null"
  }
```

```
}
```

复合指标警报状态更改时的架构

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OKActions": [String],
  "AlarmActions": [String],
  "InsufficientDataActions": [String],
  "OldStateValue": "string",
  "AlarmRule": "string",
  "TriggeringChildren": [String]
}
```

告警事件和 EventBridge

每当 CloudWatch 告警被创建、更新、删除或更改告警状态时，CloudWatch 都会将事件发送到 Amazon EventBridge。您可以使用 EventBridge 和这些事件来编写规则，这些规则会在告警更改状态时采取措施，例如通知您。有关更多信息，请参阅什么是 Amazon EventBridge？

CloudWatch 保证向 EventBridge 传送告警状态更改事件。

来自 CloudWatch 的示例事件

本部分包括来自 CloudWatch 的示例事件。

单指标告警的状态更改

```
{
    "version": "0",
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:04:40Z",
    "region": "us-east-1",
```

```
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "configuration": {
            "description": "Goes into alarm when server CPU utilization is too high!",
            "metrics": [
                {
                    "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "CPUUtilization",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ]
        },
        "previousState": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0)
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-01T13:56:40.985+0000\",\"startDate\":\"2019-10-01T13:46:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[0.0666851903306472],
\"threshold\":50.0}",
            "timestamp": "2019-10-01T13:56:40.987+0000",
            "value": "OK"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [99.50160229693434 (02/10/19 16:59:00)] was greater than the threshold (50.0) (minimum
 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:04:40.985+0000\",\"startDate\":\"2019-10-02T16:59:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[99.50160229693434],
\"threshold\":50.0}",
```

```
            "timestamp": "2019-10-02T17:04:40.989+0000",
            "value": "ALARM"
        }
    }
}
```

数学指标告警的状态更改

```
{
    "version": "0",
    "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:20:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "configuration": {
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "metrics": [
                {
                    "expression": "SUM(METRICS())",
                    "id": "e1",
                    "label": "Total Network Traffic",
                    "returnData": true
                },
                {
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkIn",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
```

```
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkOut",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                }
            ]
        },
        "previousState": {
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2019-10-02T17:20:03.642+0000",
            "value": "INSUFFICIENT_DATA"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0
 (02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint for
 OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:20:48.551+0000\",\"startDate\":\"2019-10-02T17:10:00.000+0000\",
\"period\":300,\"recentDatapoints\":[45628.0],\"threshold\":10000.0}",
            "timestamp": "2019-10-02T17:20:48.554+0000",
            "value": "ALARM"
        }
    }
}
```

异常检测告警的状态更改

```
{
    "version": "0",
    "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
    "detail-type": "CloudWatch Alarm State Change",
```

```
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-03T16:00:04Z",
    "region": "us-east-1",
    "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU Utilization
 Anomaly"],
    "detail": {
        "alarmName": "EC2 CPU Utilization Anomaly",
        "state": {
            "value": "ALARM",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0
 (03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or
 greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK ->
 ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T16:00:04.650+0000\",\"startDate\":\"2019-10-03T15:58:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.0],\"recentLowerThresholds\":
[0.020599444741798756],\"recentUpperThresholds\":[0.3006915352732461]}",
            "timestamp": "2019-10-03T16:00:04.653+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints
 [0.166666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds
 [0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803]
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T15:59:04.670+0000\",\"startDate\":\"2019-10-03T15:57:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.166666666664241],\"recentLowerThresholds\":
[0.0206719426210418],\"recentUpperThresholds\":[0.30076870222143803]}",
            "timestamp": "2019-10-03T15:59:04.672+0000"
        },
        "configuration": {
            "description": "Goes into alarm if CPU Utilization is out of band",
            "metrics": [{
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
```

```
                    "period": 60,
                    "stat": "Average"
                },
                "returnData": true
            }, {
                "id": "ad1",
                "expression": "ANOMALY_DETECTION_BAND(m1, 0.8)",
                "label": "CPUUtilization (expected)",
                "returnData": true
            }]
        }
    }
}
```

State change for a composite alarm with a suppressor alarm（带有抑制器告警的复合告警的状态变化）

```
{
    "version": "0",
    "id": "d3dfc86d-384d-24c8-0345-9f7986db0b80",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-22T15:57:45Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "actionsSuppressedReason": "Actions suppressed by WaitPeriod",
            "value": "ALARM",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.FirstChild transitioned to ALARM
 at Friday 22 July, 2022 15:57:45 UTC",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"ALARM\",\"timestamp
\":\"2022-07-22T15:57:45.394+0000\"}}]}",
            "timestamp": "2022-07-22T15:57:45.394+0000"
        },
        "previousState": {
```

```
            "value": "OK",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.Main was created and its alarm
 rule evaluates to OK",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:TotalNetworkTrafficTooHigh\",\"state\":{\"value\":\"OK\",
\"timestamp\":\"2022-07-14T16:28:57.770+0000\"}},{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"OK\",\"timestamp\":
\"2022-07-14T16:28:54.191+0000\"}}]}",
            "timestamp": "2022-07-22T15:56:14.552+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180
        }
    }
}
```

复合告警的创建

```
{

    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:22.289+0000"
        },
        "configuration": {
```

```
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "alarmName": "ServiceAggregatedAlarm",
            "description": "Aggregated monitor for instance",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:22.289+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Creation of a composite alarm with a suppressor alarm（带有抑制器告警的复合告警的创建）

```
{
    "version": "0",
    "id": "454773e1-09f7-945b-aa2c-590af1c3f8e0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:46Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
```

```
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

指标告警的更新

```
{

    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "operation": "update",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:13.757+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 80,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "CPUUtilization",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
```

```
                    "period": 300,
                    "stat": "Average"
                },
                "returnData": true
            }
        ],
        "alarmName": "ServerCpuTooHigh",
        "description": "Goes into alarm when server CPU utilization is too high!",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:34.267+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
    },
    "previousConfiguration": {
        "evaluationPeriods": 1,
        "threshold": 70,
        "comparisonOperator": "GreaterThanThreshold",
        "treatMissingData": "ignore",
        "metrics": [
            {
                "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Average"
                },
                "returnData": true
            }
        ],
        "alarmName": "ServerCpuTooHigh",
        "description": "Goes into alarm when server CPU utilization is too high!",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:13.757+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
    }
```

```
    }
}
```

Update of a composite alarm with a suppressor alarm（带有抑制器告警的复合告警的更新）

```
{
    "version": "0",
    "id": "4c6f4177-6bd5-c0ca-9f05-b4151c54568b",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:56Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "update",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
```

```
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

指标数学告警的删除

```
{

    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:07:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "operation": "delete",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:17.672+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 10000,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [{
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkIn",
                            "dimensions": {
```

```
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Maximum"
                },
                "returnData": false
            },
            {
                "id": "m2",
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "NetworkOut",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 300,
                    "stat": "Maximum"
                },
                "returnData": false
            },
            {
                "id": "e1",
                "expression": "SUM(METRICS())",
                "label": "Total Network Traffic",
                "returnData": true
            }
        ],
        "alarmName": "TotalNetworkTrafficTooHigh",
        "description": "Goes into alarm if total network traffic exceeds 10Kb",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:17.672+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []

    }
  }
}
```

Deletion of a composite alarm with a suppressor alarm（带有抑制器告警的复合告警的删除）

```
{
    "version": "0",
    "id": "e34592a1-46c0-b316-f614-1b17a87be9dc",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T14:00:01Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "delete",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```


管理警报

编辑或删除 CloudWatch 警报

您可以编辑或删除现有告警。

您无法更改现有告警的名称。您可以复制一个告警，并为新告警指定不同的名称。要复制一个告警，请在告警列表中选中该告警名称旁边的复选框，然后选择 Action（操作）和 Copy（复制）。

编辑告警

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。
3. 选择警报的名称。
4. 要添加或移除标签，请选择标签选项卡，然后选择管理标签。
5. 要编辑告警的其他部分，请依次选择操作、编辑。

   将显示 Specify metric and conditions（指定指标和条件）页面，其中显示一个图表以及有关您选择的指标和统计数据的其他信息。
6. 要更改指标，请选择 Edit（编辑），选择 All metrics（全部指标）选项卡，然后执行以下操作之一：
   - 选择包含所需指标的服务命名空间。继续选择所显示的选项，以缩小选择范围。在显示指标列表时，选中所需的指标旁边的复选框。
   - 在搜索框中，输入指标名称、维度或资源 ID，然后按 Enter。接下来，选择其中的一个结果并继续，直到显示指标列表。选中所需的指标旁边的复选框。

   选择选择指标。
7. 要更改告警的其他内容，请选择相应的选项。要更改必须有多少个数据点违例以使告警变为 ALARM（告警）状态或更改缺失数据的处理方式，请选择 Additional configuration（其他配置）。
8. 选择下一步。
9. 在 Notification（通知）、Auto Scaling action（Auto Scaling 操作）和 EC2 action（EC2 操作）下，编辑在触发告警时执行的操作（可选）。然后选择下一步。
10. （可选）更改告警描述。

    您无法更改现有告警的名称。您可以复制一个告警，并为新告警指定不同的名称。要复制一个告警，请在告警列表中选中该告警名称旁边的复选框，然后选择 Action（操作）和 Copy（复制）。
11. 选择下一步。
12. 在 Preview and create（预览和创建）下，确认具有所需的信息和条件，然后选择 Update alarm（更新告警）。

更新使用 Amazon SNS 控制台创建的电子邮件通知列表

1. 通过 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在导航窗格中，选择 Topics（主题），然后选择通知列表（主题）的 ARN。
3. 请执行以下操作之一：
   - 要添加电子邮件地址，请选择 Create subscription（创建订阅）。对于协议，选择电子邮件。对于 Endpoint（端点），输入新收件人的电子邮件地址。选择 Create subscription（创建订阅）。
   - 要删除电子邮件地址，请选择 Subscription ID（订阅 ID）。选择 Other subscription actions（其他订阅操作）、Delete subscriptions（删除订阅）。
4. 选择 Publish to topic（发布到主题）。

删除警报

1. 打开 CloudWatch 控制台，网址为：https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择警报。
3. 选中告警名称左侧的复选框，然后依次选择 Actions（操作）和 Delete（删除）。
4. 选择删除。

隐藏 Auto Scaling 警报

在 AWS Management Console 中查看您的警报时，您可以隐藏与 Amazon EC2 Auto Scaling 相关的警报。仅在 AWS Management Console 中提供了该功能。

若要临时隐藏 Auto Scaling 警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警），然后选择 Hide Auto Scaling alarms（隐藏弹性伸缩告警）。

警报使用场景和示例

以下章节提供了常见使用场景的警报示例和教程。

创建账单告警以监控预估的 AWS 费用

您可以使用 Amazon CloudWatch 监控 AWS 估算费用。在为您的 AWS 账户启用估算费用监控时，将每天计算几次估算费用并作为指标数据发送到 CloudWatch。

账单指标数据存储在美国东部（弗吉尼亚北部）区域中，并表示全球费用。该数据包括您在 AWS 中使用的每个服务的估算费用，以及您的 AWS 估算费用总和。

当您的账户账单超过阈值时，将触发告警。它仅在当前账单超出阈值时触发。它不会根据您本月迄今为止的使用情况进行预测。

如果您每当费用超出阈值时创建一个账单告警，则告警将立即变为 ALARM（告警）状态。

> **Note**
>
> 有关分析已计费的 CloudWatch 费用的信息，请参阅 CloudWatch 账单和成本。

任务

- 启用账单提醒
- 创建账单警报
- 删除账单告警

启用账单提醒

在为估算费用创建告警之前，您必须启用账单提醒，以便监控 AWS 估算费用并使用账单指标数据创建告警。在启用账单提醒后，您无法禁用数据收集，但可以删除创建的任何账单告警。

首次启用账单提醒后，大约需要 15 分钟时间，您就可以查看账单数据和设置账单告警。

要求

- 您必须使用账户根用户凭证或作为被授予权限的 IAM 用户登录，才能查看账单信息。
- 对于整合账单账户，每个关联账户的账单数据可以在付款账户登录后找到。您可以查看每个关联账户以及整合账户的估计费用总和，和各项服务的估计费用。
- 在整合账单账户中，仅当付款人账户启用 Receive Billing Alerts（接收账单提醒）首选项时，才会捕获成员关联账户的指标。如果您更改了您的管理账户/付款人账户，则必须在新的管理账户/付款人账户中启用账单提醒。

- 该账户不能属于 Amazon 合作伙伴网络 (APN)，因为对于 APN 账户，账单指标不会发布到 CloudWatch。有关更多信息，请参阅 AWS 合作伙伴网络。

要启用预估收费监控

1. 打开 AWS Billing 控制台，网址为：https://console.aws.amazon.com/billing/。
2. 在导航窗格中，选择 Billing Preferences（账单首选项）。
3. 通过提醒首选项选择编辑。
4. 选择接收 CloudWatch 账单提醒。
5. 选择保存首选项。

创建账单警报

> **Important**
>
> 创建账单告警之前，您必须将区域设置为美国东部（弗吉尼亚州北部）。账单指标数据存储在该区域中，并表示全球费用。您还必须为您的账户启用账单提醒；或者，如果您使用的是整合账单，则必须在管理账户/付款人账户中启用账单提醒。有关更多信息，请参阅 启用账单提醒。

在该过程中，您可以创建一个告警，以便在 AWS 的估算费用超出定义的阈值时发送通知。

使用 CloudWatch 控制台创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Alarms（告警），然后选择 All alarms（所有告警）。
3. 选择创建警报。
4. 选择选择指标。在 Browse（浏览）中，选择 Billing（账单），然后选择 Total Estimated Charge（总估算费用）。

> **Note**
>
> 如果您没看到账单/总估算费用指标，则启用账单提醒，并将您的区域更改为美国东部（弗吉尼亚州北部）。有关更多信息，请参阅 启用账单提醒。

5. 选中 EstimatedCharges（估算费用）指标的复选框，然后选择 Select metric（选择指标）。
6. 对于 Statistic（统计数据），选择 Maximum（最大）。
7. 对于 Period（周期），选择 6 hours（6 小时）。
8. 对于阈值类型，选择静态。
9. 对于 Whenever EstimatedCharges is . . .（当估算费用. .），选择 Greater（大）。
10. 对于 than . . .，请定义要触发告警的值。例如，**200** USD。

    EstimatedCharges 指标值仅以美元（USD）为单位，货币转换由 Amazon Services LLC 提供。有关更多信息，请参阅什么是 AWS Billing？。

    > **Note**
    >
    > 定义阈值后，预览图显示您当月的预估费用。

11. 在其他配置中，执行以下操作：
    - 对于 Datapoints to alarm（触发告警的数据点数），指定 1 out of 1（1 选 1）。
    - 对于 Missing data treatment（缺失数据处理），选择 Treat missing data as missing（将缺失的数据视为缺失）。
12. 选择下一步。
13. 在通知下，确保选择告警中。选择当您的告警处于 ALARM 状态时要通知的 Amazon SNS 主题。Amazon SNS 主题可以包含您的电子邮件地址，这样当账单金额超过您指定的阈值时，您就可以收到电子邮件。

    您可以选择现有的 Amazon SNS 主题、创建一个新 Amazon SNS 主题或使用主题 ARN 通知其他账户。如果您希望您的告警为相同告警状态或不同告警状态发送多个通知，请选择 Add notification（添加通知）。
14. 选择下一步。
15. 在 Name and description（名称和描述）下，为您的告警输入名称。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。
    - （可选）输入告警的描述。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。
16. 选择下一步。

17. 在 Preview and create（预览和创建）下，确保您的配置正确，然后选择 Create alarm（创建告警）。

删除账单告警

当您不再需要账单告警时，可将其删除。

删除账单告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 如果需要，可将区域更改为美国东部（弗吉尼亚北部）。账单指标数据存储在此区域中，并且反映全球费用。
3. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。
4. 选中告警旁的复选框，然后依次选择 Actions（操作） 和 Delete（删除）。
5. 当系统提示进行确认时，选择 Yes, Delete（是，删除）。

创建 CPU 使用率告警

您可以创建在告警状态从 OK（正常）变为 ALARM（告警）时使用 Amazon SNS 发送通知的 CloudWatch 告警。

在 EC2 实例的平均 CPU 使用率在指定的连续评估期内超出指定的阈值时，告警将变为 ALARM（告警）状态。

使用 AWS Management Console 设置 CPU 使用率告警

可以执行以下步骤以使用 AWS Management Console创建 CPU 使用率警报。

根据 CPU 利用率创建警报

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。
3. 选择创建警报。
4. 选择选择指标。
5. 在 All metrics（所有指标）选项卡中，选择 EC2 metrics（EC2 指标）。
6. 选择一个指标类别（例如，Per-Instance Metrics（每个实例的指标））。

7. 找到您希望在 InstanceId（实例 ID）列和 Metric Name（指标名称）中的 CPUUtilization（CPU 使用率）中列出的实例所在的行。选中此行旁边的复选框，然后选择 Select metric（选择指标）。
8. 在 Specify metric and conditions（指定指标和条件）下，对于 Statistic（统计数据），选择 Average（平均值），然后选择一个预定义百分位数，或指定自定义百分位数（例如 **p95.45**）。
9. 选择时间段（例如 **5 minutes**）。
10. 在条件下面，指定以下内容：

    a. 对于 Threshold type（阈值类型），选择 Static（静态）。

    b. 对于 Whenever CPUUtilization is（每当 CPU 利用率），指定 Greater（大）。在 than...（于...）下，指定在 CPU 使用率超过此百分比时触发告警进入“ALARM（告警）”状态的阈值。例如，70。

    c. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

    要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。

    d. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。

    e. 如果警报将百分比值作为监控的统计数据，将显示样本数少的百分比框。使用它来选择是评估还是忽略采样率低的案例。如果选择忽略 (保持警报状态)，在样本大小太小时，将始终保持当前警报状态。有关更多信息，请参阅 基于百分位数的 CloudWatch 告警和小数据样本。
11. 选择下一步。
12. 在 Notification（通知）下，选择 In alarm（处于告警中），并选择当告警处于 ALARM 状态时要通知的 SNS 主题

    要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

    要让警报不发送通知，请选择删除。
13. 在完成后，选择下一步。
14. 输入警报的名称和说明。然后选择下一步。

    名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。

15. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

使用 AWS CLI 设置 CPU 使用率告警

可以执行以下步骤以使用 AWS CLI创建 CPU 使用率警报。

根据 CPU 利用率创建警报

1. 设置 SNS 主题。有关更多信息，请参阅 设置 Amazon SNS 通知。
2. 使用 put-metric-alarm 命令创建告警，如下所示。

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic
 Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --
dimensions  Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions
 arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. 通过使用 set-alarm-state 命令强制更改警报状态来测试警报。

   a. 将告警状态从 INSUFFICIENT_DATA（数据不足）更改为 OK（正常）。

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value OK
   ```

   b. 将告警状态从 OK（正常）更改为 ALARM（告警）。

   ```
   aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. 检查您是否收到有关告警的电子邮件通知。

创建发送电子邮件的负载均衡器延迟告警

您可以设置 Amazon SNS 通知并配置告警，此告警监控经典负载均衡器超过 100 毫秒的延迟。

使用 AWS Management Console 设置延迟告警

可以执行以下步骤以使用 AWS Management Console创建负载均衡器延迟警报。

创建负载均衡器延迟告警

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。
3. 选择Create alarm（创建警报）。
4. 在 CloudWatch Metrics by Category（按类别显示的 CloudWatch 指标）下，选择 ELB Metrics（ELB 指标）类别。
5. 选择包含经典负载均衡器和 Latency（延迟）指标的行。
6. 对于统计数据，选择 Average（平均值），然后选择其中的一个预定义百分比值，或者指定一个自定义百分比值（例如 **p95.45**）。
7. 对于时间段，选择 1 Minute（1 分钟）。
8. 选择下一步。
9. 在警报阈值下面，输入警报的唯一名称（例如，**myHighCpuAlarm**）和警报描述（例如，**Alarm when Latency exceeds 100s**）。告警名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符

   名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。
10. 在 Whenever（每当）下，对于 is（是），选择 **>** 并输入 **0.1**。对于 for（持续时间），输入 **3**。
11. 在 Additional settings（附加设置）下，对于 Treat missing data as（将缺失的数据作为以下内容处理）选择 ignore (maintain alarm)（忽略（保持告警状态）），以使缺失数据点不会触发告警状态更改。

    对于 Percentiles with low samples（样本数少的百分比），选择 ignore (maintain the alarm state)（忽略（保持告警状态）），使告警只评估具有充足数量的数据样本的情况。
12. 在 Actions（操作）下，为 Whenever this alarm（每当此告警）选择 State is ALARM（状态为“告警”）。对于 Send notification to（发送通知到），选择一个现有 SNS 主题或创建一个新 SNS 主题。

    要创建 SNS 主题，请选择 New list（新列表）。对于 Send notification to（发送通知到）输入 SNS 主题的名称（例如 **myHighCpuAlarm**），对于 Email list（电子邮件列表）输入在告警状态变为 ALARM（告警）时通知的电子邮件地址列表（以逗号分隔）。将向每个电子邮件地址发送一封主题订阅确认电子邮件。您必须先确认订阅，然后才会发送通知。
13. 选择创建警报。

使用 AWS CLI 设置延迟告警

可以执行以下步骤以使用 AWS CLI创建负载均衡器延迟警报。

创建负载均衡器延迟告警

1. 设置 SNS 主题。有关更多信息，请参阅 设置 Amazon SNS 通知。
2. 使用 put-metric-alarm 命令创建告警，如下所示：

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm
 when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic
 Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --
dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-
actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. 通过使用 set-alarm-state 命令强制更改警报状态来测试警报。

   a. 将告警状态从 INSUFFICIENT_DATA（数据不足）更改为 OK（正常）。

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value OK
   ```

   b. 将告警状态从 OK（正常）更改为 ALARM（告警）。

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. 检查您是否收到有关告警的电子邮件通知。

创建发送电子邮件的存储吞吐量告警

您可以设置 SNS 通知并配置告警，该告警在 Amazon EBS 超过 100MB 吞吐量时触发。

使用 AWS Management Console 设置存储吞吐量告警

可以执行以下步骤，使用 AWS Management Console 创建基于 Amazon EBS 吞吐量的告警。

创建存储吞吐量告警

1. 访问 https://console.aws.amazon.com/cloudwatch/，打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。

3. 选择Create alarm（创建警报）。
4. 在 EBS Metrics（EBS 指标）下，选择一个指标类别。
5. 选择包含卷和 VolumeWriteBytes 指标的行。
6. 对于统计数据，选择 Average（平均值）。对于时间段，选择 5 Minutes（5 分钟）。选择下一步。
7. 在警报阈值下面，输入警报的唯一名称（例如，**myHighWriteAlarm**）和警报描述（例如，**VolumeWriteBytes exceeds 100,000 KiB/s**）。名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。
8. 在 Whenever（每当）下，对于 is（是），选择 **>** 并输入 **100000**。对于 for（持续时间），输入 **15** 个连续评估期。

   Alarm Preview（告警预览）下会显示阈值的图形表示。
9. 在 Additional settings（附加设置）下，对于 Treat missing data as（将缺失的数据作为以下内容处理）选择 ignore (maintain alarm)（忽略（保持告警状态）），以使缺失数据点不会触发告警状态更改。
10. 在 Actions（操作）下，为 Whenever this alarm（每当此告警）选择 State is ALARM（状态为“告警”）。对于 Send notification to（发送通知到），选择现有的 SNS 主题或创建一个主题。

    要创建 SNS 主题，请选择 New list（新列表）。对于 Send notification to（发送通知到）输入 SNS 主题的名称（例如 **myHighCpuAlarm**），对于 Email list（电子邮件列表）输入在告警状态变为 ALARM（告警）时通知的电子邮件地址列表（以逗号分隔）。将向每个电子邮件地址发送一封主题订阅确认电子邮件。您必须先确认订阅，然后才能将通知发送到电子邮件地址。
11. 选择 Create Alarm（创建告警）。

使用 AWS CLI 设置存储吞吐量告警

可以执行以下步骤，使用 AWS CLI 创建基于 Amazon EBS 吞吐量的告警。

创建存储吞吐量告警

1. 创建 SNS 主题。有关更多信息，请参阅 设置 Amazon SNS 通知。
2. 创建告警。

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm
 when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --
```

```
namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --
comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-
volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-
east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-
east-1:111122223333:my-insufficient-data-topic
```

3. 通过使用 set-alarm-state 命令强制更改警报状态来测试警报。

   a. 将告警状态从 INSUFFICIENT_DATA（数据不足）更改为 OK（正常）。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value OK
   ```

   b. 将告警状态从 OK（数据不足）更改为 ALARM（正常）。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. 将告警状态从 ALARM（正常）更改为 INSUFFICIENT_DATA（告警）。

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value INSUFFICIENT_DATA
   ```

   d. 检查您是否收到有关告警的电子邮件通知。

针对 AWS 数据库中的性能详情计数器指标创建警报

CloudWatch 包含 DB_PERF_INSIGHTS 指标数学函数，您可以使用该函数将 Amazon Relational Database Service 和 Amazon DocumentDB（与 MongoDB 兼容）中的性能详情计数器指标引入 CloudWatch。DB_PERF_INSIGHTS 还以亚分钟为间隔引入 DBLoad 指标。您可以根据这些指标设置 CloudWatch 警报。

有关 Amazon RDS 性能详情的更多信息，请参阅在 Amazon RDS 上使用性能详情监控数据库负载。

有关 Amazon DocumentDB 性能详情的更多信息，请参阅使用性能详情进行监控。

基于 DB_PERF_INSIGHTS 函数的警报不支持异常检测。

> **Note**
>
> DB_PERF_INSIGHTS 检索的精度为亚分钟的高分辨率指标仅适用于 DBLoad 指标，或者如果您启用了更高分辨率的增强监控，则适用于操作系统指标。有关 Amazon RDS 增强监控的更多信息，请参阅使用增强监控监控操作系统指标。
> 您可以使用 DB_PERF_INSIGHTS 函数创建高分辨率警报。高分辨率警报的最长评估范围为三小时。您可以使用 CloudWatch 控制台绘制任何时间范围内通过 DB_PERF_INSIGHTS 函数检索到的指标的图表。

创建基于性能详情指标的警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Alarms（告警），然后选择 All alarms（所有告警）。
3. 选择创建警报。
4. 选择 Select Metric（选择指标）。
5. 选择添加数学下拉列表，然后从列表中选择所有函数、DB_PERF_INSIGHTS。

   选择 DB_PERF_INSIGHTST 后，系统将显示一个数学表达式框，您可以在其中应用或编辑数学表达式。
6. 在数学表达式框中，输入 DB_PERF_INSIGHTS 数学表达式，然后选择应用。

   例如，**DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', 'os.cpuUtilization.user.avg')**

   > **Important**
   >
   > 使用 DB_PERF_INSIGHTS 数学表达式时，必须指定数据库的唯一数据库资源 ID。这与数据库标识符不同。要在 Amazon RDS 控制台中查找数据库资源 ID，请选择数据库实例以查看其详细信息。然后，选择配置选项卡。资源 ID 将显示在配置部分中。

   有关 DB_PERF_INSIGHTS 函数和可用于指标数学的其他函数的信息，请参阅 指标数学语法和函数。
7. 选择选择指标。

将显示 Specify metric and conditions（指定指标和条件）页面，其中显示一个图表以及有关您选择的数学表达式的其他信息。

8. 对于 Whenever ***expression*** is（每当表达式），指定表达式是必须大于、小于还是等于阈值。在于... 下面，指定阈值。
9. 选择其他配置。对于触发警报的数据点数，指定必须有多少个评估期（数据点）处于 ALARM 状态才能触发警报。如果此处的两个值匹配，则会创建一个告警；如果多个连续评估期违例，该告警将变为 ALARM（告警）状态。

   要创建“M（最大为 N）”告警，为第一个值指定的数字应小于为第二个值指定的数字。有关更多信息，请参阅 评估告警。
10. 对于缺失数据处理，选择在缺失某些数据点时的警报行为。有关更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。
11. 选择下一步。
12. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。

    要使告警为相同告警状态或不同告警状态发送多个通知，请选择添加通知。

    要让警报不发送通知，请选择删除。
13. 要让警报执行 Auto Scaling、EC2、Lambda 或 Systems Manager 操作，请选择相应的按钮，然后选择警报状态和要执行的操作。如果您选择 Lambda 函数作为警报操作，则需要指定函数名称或 ARN，并且可以选择该函数的特定版本。

    告警只有在进入“ALARM（告警）”状态时才能执行 Systems Manager 操作。有关 Systems Manager 操作的更多信息，请参阅将 CloudWatch 配置为通过告警创建 OpsItems 和事件创建。

    > **Note**
    >
    > 要创建执行 SSM Incident Manager 操作的告警，您必须具有特定的权限。有关更多信息，请参阅 AWS Systems Manager Incident Manager 的基于身份的策略示例。
14. 在完成后，选择下一步。
15. 输入警报的名称和说明。然后选择下一步。

    名称必须仅包含 UTF-8 字符，并且不能包含 ASCII 控制字符。描述可以包含 Markdown 格式，该格式仅在 CloudWatch 控制台的警报详细信息选项卡中显示。Markdown 非常适合用于向运行手册或其他内部资源添加链接。

16. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

创建告警以停止、终止、重启或恢复 EC2 实例

利用 Amazon CloudWatch 告警操作，您可创建自动停止、终止、重启或恢复 EC2 实例的告警。当不再需要某个实例运行时，您可使用停止或终止操作来帮助您节省资金。如果发生了系统损害，您可使用重启和恢复操作自动重启这些实例或将它们恢复到新硬件上。

在许多情况下，您可能需要自动终止或停止实例。例如，您可能拥有专用于批工资单处理作业或科学计算任务的实例，这些实例在运行一段时间后就完成了其工作。与其让这些实例空闲（并产生费用），不如将其停止或终止以帮助节省开支。使用停止警报操作和终止警报操作的主要区别是，停止的警报可以在以后需要运行时轻松重启。您还可以保留相同的实例 ID 和根卷。而终止的实例则无法重新启动。如此就必须启动一个新的实例。

您可以向为 Amazon EC2 每个实例指标设置的任何告警添加停止、终止或重启操作，这些指标包括 Amazon CloudWatch 提供的基本和详细监控指标（在亚马逊云科技/EC2 命名空间中），以及包含“InstanceId=”维度的任何自定义指标，只要 InstanceId 值引用有效运行的 Amazon EC2 实例。您还可以将恢复操作添加到针对任何 Amazon EC2 每个实例指标设置的告警中，但以下情况除外：`StatusCheckFailed_Instance`。

> **Important**
>
> 如果缺失指标数据点，则在 Amazon EC2 指标上配置的警报可能会暂时进入 INSUFFICIENT_DATA 状态。这种情况很少见，但在指标报告中断时可能会发生，即使在 Amazon EC2 实例运行正常的情况下也是如此。对于配置为采取停止、终止、重启或恢复操作的 Amazon EC2 指标的警报，我们建议您将这些警报配置为将缺失的数据视为 `missing`，并使这些警报仅在处于 ALARM 状态时被触发。
>
> 有关如何配置 CloudWatch 对已设置警报的缺失指标执行操作的更多信息，请参阅 配置 CloudWatch 告警处理缺失数据的方式。

要设置可重启、停止或终止实例的 CloudWatch 告警操作，您必须使用服务相关的 IAM 角色 AWSServiceRoleForCloudWatchEvents。AWSServiceRoleForCloudWatchEvents IAM 角色允许 AWS 代表您执行告警操作。

要为 CloudWatch Events 创建服务相关角色，请使用以下命令：

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

控制台支持

您可使用 CloudWatch 控制台或 Amazon EC2 控制台创建告警。本文档中的程序使用 CloudWatch 控制台。有关使用 Amazon EC2 控制台的程序，请参阅《Amazon EC2 用户指南》中的创建停止、终止、重启或恢复实例的警报。

权限

如果您使用 AWS Identity and Access Management (IAM) 账户来创建或修改执行 EC2 操作或 Systems Manager OpsItem 操作的告警，您必须拥有 `iam:CreateServiceLinkedRole` 权限。

内容

- 在 Amazon CloudWatch 告警中添加停止操作
- 在 Amazon CloudWatch 告警中添加终止操作
- 在 Amazon CloudWatch 告警中添加重启操作
- 在 Amazon CloudWatch 告警中添加恢复操作
- 查看已触发的告警和操作的历史记录

在 Amazon CloudWatch 告警中添加停止操作

可以创建当达到一定阈值后停止 Amazon EC2 实例的警报。例如，您可能运行了开发或测试实例而偶尔忘记将其关闭。可以创建当平均 CPU 使用率低于 10% 达 24 小时时触发的警报，同时告知其为空闲并不再使用。可以根据需要调整阈值、时长和时间段，还可以添加 SNS 通知，以便您在触发警报后能够收到电子邮件。

可以停止或终止将 Amazon Elastic Block Store 卷用作根设备的 Amazon EC2 实例，但只能终止将实例存储用作根设备的实例。

使用 Amazon CloudWatch 控制台创建停止空闲实例的告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。
3. 选择创建警报。
4. 选择 Select Metric（选择指标）。
5. 对于 AWS 命名空间，请选择 EC2。
6. 执行以下操作：

    a. 选择 Per-Instance Metrics（每个实例的指标）。

b. 选择包含正确实例和 CPUUtilization 指标的行中的复选框。
c. 选择绘成图表的指标选项卡。
d. 对于统计数据，选择 Average（平均值）。
e. 选择时间段（例如 **1 Hour**）。
f. 选择选择指标。

7. 对于 Define Alarm 步骤，执行以下操作：

a. 在 Conditions（条件）下，选择 Static（静态）。
b. 在 Whenever CPUUtilization is（每当 CPUUtilization）下，选择 Lower（降低）。
c. 对于 than（比较），键入 **10**。
d. 选择下一步。
e. 在 Notification 下，为 Send notification to 选择一个现有 SNS 主题或创建一个新 SNS 主题。

   要创建 SNS 主题，请选择 New list（新列表）。对于发送通知到，键入 SNS 主题的名称（例如“Stop_EC2_Instance”）。对于 Email list (电子邮件列表)，请键入警报变为 ALARM 状态时将通知发送到的电子邮件地址列表（以逗号分隔）。将向每个电子邮件地址发送一封主题订阅确认电子邮件。您必须先确认订阅，然后才能将通知发送到电子邮件地址。
f. 选择 Add EC2 Action（添加 EC2 操作）。
g. 对于 Alarm state trigger（告警状态触发器），选择 In alarm（在告警中）。对于 Take the following action（请执行以下操作），选择 Stop this instance（停止此实例）。
h. 选择下一步。
i. 输入警报的名称和说明。名称只能包含 ASCII 字符。然后选择下一步。
j. 在 Preview and create 下面，确认具有所需的信息和条件，然后选择 Create alarm。

在 Amazon CloudWatch 告警中添加终止操作

可以创建当达到一定阈值时自动终止 EC2 实例的警报 (只要该实例未启用终止保护)。例如，某个实例已经完成工作，您不再需要此实例而希望将其终止。如果可能在之后使用该实例，则应该选择停止而不是终止。有关为实例启用和禁用终止保护的信息，请参阅《Amazon EC2 用户指南》中的为实例启用终止保护。

使用 Amazon CloudWatch 控制台创建终止空闲实例的告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms 和 Create Alarm。

3. 对于 Select Metric 步骤，执行以下操作：

   a. 在 EC2 Metrics 下，选择 Per-Instance Metrics。

   b. 选择包含实例和 CPUUtilization 指标的行。

   c. 对于统计数据，选择 Average（平均值）。

   d. 选择时间段（例如 **1 Hour**）。

   e. 选择下一步。

4. 对于 Define Alarm 步骤，执行以下操作：

   a. 在 Alarm Threshold 下，键入警报的唯一名称 (例如“Terminate EC2 instance”) 和警报的描述 (例如“Terminate EC2 instance when CPU is idle for too long”)。警报名称必须仅包含 ASCII 字符。

   b. 在每当下，为 是选择 **<** 并键入 **10**。对于对于，键入 **24** 作为连续时段数。

      Alarm Preview（告警预览）下会显示阈值的图形表示。

   c. 在 Notification 下，为 Send notification to 选择一个现有 SNS 主题或创建一个新 SNS 主题。

      要创建 SNS 主题，请选择 New list（新列表）。对于发送通知到，键入 SNS 主题的名称（例如“Terminate_EC2_Instance”）。对于 Email list (电子邮件列表)，请键入警报变为 ALARM 状态时将通知发送到的电子邮件地址列表（以逗号分隔）。将向每个电子邮件地址发送一封主题订阅确认电子邮件。您必须先确认订阅，然后才能将通知发送到电子邮件地址。

   d. 选择 EC2 Action。

   e. 对于每当此警报，请选择状态为“警报”。对于 Take this action，选择 Terminate this instance。

   f. 选择创建警报。

在 Amazon CloudWatch 告警中添加重启操作

您可创建监控 Amazon EC2 实例并自动重启此实例的 Amazon CloudWatch 警报。在实例运行状况检查失败时，推荐重启警报操作（与恢复警报操作相反，该操作适合系统运行状况检查失败的情况）。实例重启相当于操作系统重启。在许多情况下，只需要几分钟时间即可重启您的实例。重启实例时，其仍驻留在相同的物理主机上，因此您的实例将保留其公有 DNS 名称、私有 IP 地址及其实例存储卷上的任何数据。

重启实例不会启动新的实例计费时间，这与停止并重新启动您的实例不同。有关重新启动实例的更多信息，请参阅《Amazon EC2 用户指南》中的重启实例。

> **Important**
>
> 为了避免重启操作与恢复操作之间的竞争情况，请避免为重启警报和恢复警报设置相同的评估期。我们建议您将重启警报设置为 3 个 1 分钟的评估期。

使用 Amazon CloudWatch 控制台创建重启实例的告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms 和 Create Alarm。
3. 对于 Select Metric 步骤，执行以下操作：
   a. 在 EC2 Metrics 下，选择 Per-Instance Metrics。
   b. 选择包含实例和 StatusCheckFailed_Instance 指标的行。
   c. 对于统计数据，选择 Minimum。
   d. 选择时间段（例如 **1 Minute**）。
   e. 选择下一步。
4. 对于 Define Alarm 步骤，执行以下操作：
   a. 在 Alarm Threshold 下，键入警报的唯一名称 (例如“Reboot EC2 instance”) 和警报的描述 (例如“Reboot EC2 instance when health checks fail”)。警报名称必须仅包含 ASCII 字符。
   b. 在每当下，为是选择 **>** 并键入 **0**。对于对于，键入 **3** 作为连续时段数。

      Alarm Preview（告警预览）下会显示阈值的图形表示。
   c. 在 Notification 下，为 Send notification to 选择一个现有 SNS 主题或创建一个新 SNS 主题。

      要创建 SNS 主题，请选择 New list（新列表）。对于发送通知到，键入 SNS 主题的名称（例如“Reboot_EC2_Instance”）。对于 Email list (电子邮件列表)，请键入警报变为 ALARM 状态时将通知发送到的电子邮件地址列表（以逗号分隔）。将向每个电子邮件地址发送一封主题订阅确认电子邮件。您必须先确认订阅，然后才能将通知发送到电子邮件地址。
   d. 选择 EC2 Action。
   e. 对于每当此警报，请选择状态为“警报”。对于 Take this action，选择 Reboot this instance。
   f. 选择创建警报。

在 Amazon CloudWatch 告警中添加恢复操作

您可以创建 Amazon CloudWatch 警报用于监控 Amazon EC2 实例，并且在实例受损（由于发生底层硬件故障或需要 AWS 参与才能修复的问题）时自动恢复实例。无法恢复终止的实例。恢复的实例与原始实例相同，包括实例 ID、私有 IP 地址、弹性 IP 地址以及所有实例元数据。

当 StatusCheckFailed_System 告警触发且恢复操作启动时，您在创建告警及相关恢复操作时所选择的 Amazon SNS 主题将向您发出通知。在实例恢复过程中，实例将在重启时迁移，并且内存中的所有数据都将丢失。当该过程完成后，会向您已配置警报的 SNS 主题发布信息。任何订阅此 SNS 主题的用户都将收到一封电子邮件通知，其中包括恢复尝试的状态以及任何进一步的指示。您会注意到，实例在已恢复的实例上重启。

恢复操作仅适用于 StatusCheckFailed_System，而不能用于 StatusCheckFailed_Instance。

导致系统状态检查出现故障的问题示例包括：

- 网络连接丢失
- 系统电源损耗
- 物理主机上的软件问题
- 物理主机上影响到网络连接状态的硬件问题

只有某些实例类型支持恢复操作。有关支持的实例类型和其他要求的更多信息，请参阅恢复实例和要求。

> **Important**
>
> 为了避免重启操作与恢复操作之间的竞争情况，请避免为重启警报和恢复警报设置相同的评估期。我们建议您将恢复警报设置为 2 个 1 分钟的评估期，并将重启警报设置为 3 个 1 分钟的评估期。

使用 Amazon CloudWatch 控制台创建恢复实例的告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Alarms（警报）和 All alarms（所有警报）。
3. 选择创建警报。
4. 选择选择指标并执行以下操作：

a. 选择EC2 指标、每个实例的指标。
b. 选择包含实例和 StatusCheckFailed_System 指标的行，然后选择选择指标。
c. 对于统计数据，选择 Minimum。
d. 选择时间段（例如 **1 Minute**）。

> **Important**
>
> 为了避免重启操作与恢复操作之间的竞争情况，请避免为重启警报和恢复警报设置相同的评估期。我们建议您将恢复警报设置为 2 个 1 分钟的评估期。

5. 对于条件，执行以下操作：
   a. 在阈值类型下，选择静态。
   b. 在每当下，选择大于，然后对于比... 输入 **0**。
   c. 选择其他配置，然后在待警报的数据点中指定 2 个（共 2 个）。
6. 选择下一步。
7. 在通知下，执行以下操作：
   a. 对于 Alarm state trigger（告警状态触发器），选择 In alarm（在告警中）。
   b. 对于发送通知到如下 SNS 主题，选择一个现有 SNS 主题或创建一个新 SNS 主题。
   c. 选择 Add EC2 Action（添加 EC2 操作）。
   d. 对于 Alarm state trigger（告警状态触发器），选择 In alarm（在告警中）。
   e. 对于请执行以下操作，选择恢复此实例。
   f. 选择下一步。
8. 对于警报名称，输入警报的唯一名称（例如，**Recover EC2 instance**）和警报描述（例如，**Recover EC2 instance when health checks fail**）。警报名称必须仅包含 ASCII 字符。
9. 选择下一步。
10. 选择创建警报。

查看已触发的告警和操作的历史记录

您可以在 Amazon CloudWatch 控制台中查看警报和操作历史记录。Amazon CloudWatch 会保留最近 30 天的警报和操作历史记录。

要查看已触发的警报和操作的历史记录

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Alarms (警报)，然后选择一个警报。
3. 要查看最近的状态转换以及时间和指标值，请选择详细信息。
4. 要查看最近的历史记录条目，请选择 History (历史记录)。

警报和标记

标签是键值对，可以帮助您对资源进行组织和分类。您还可以使用它们来限定用户权限的范围，方法是只授予用户访问或更改具有特定标签值的资源的权限。有关标记资源的更多一般信息，请参阅标记 AWS 资源

以下列表说明了标记如何与 CloudWatch 警报配合使用的一些详细信息。

- 要能够设置或更新 CloudWatch 资源的标签，您必须登录拥有 `cloudwatch:TagResource` 权限的账户。例如，要创建警报并为其设置标签，除了 `cloudwatch:TagResource` 权限之外，您还必须拥有 the `cloudwatch:PutMetricAlarm` 权限。建议您确保组织中要创建或更新 CloudWatch 资源的任何人都拥有 `cloudwatch:TagResource` 权限。
- 标签可用于基于标签的授权控制。例如，IAM 用户或角色权限可以包含一些条件，根据其标签将 CloudWatch 调用限制为特定的资源。不过，请注意以下事项
  - 名称以 `aws:` 开头的标签不能用于基于标签的授权控制。
  - 复合警报不支持基于标签的授权控制。

应用程序监控（APM）

CloudWatch 包括以下可观测性解决方案，可用于帮助您监控自己的应用程序和服务。

主题

- Application Signals
- 服务级别目标（SLO）
- 综合监控（Canary）
- CloudWatch RUM
- 使用 CloudWatch Evidently 执行启动和 A/B 实验

Application Signals

使用 CloudWatch Application Signals 自动检测 AWS 上的应用程序，以监控当前应用程序的运行状况，并根据业务目标跟踪长期应用程序性能。Application Signals 为您提供统一的、以应用程序为中心的应用程序、服务和依赖项视图，帮助您监控应用程序的运行状况并对其进行分类。

- 启用 Application Signals 以自动从您的应用程序收集指标和跟踪，并显示关键指标，例如调用量、可用性、延迟、故障和错误。无需编写自定义代码或创建控制面板，即可快速查看当前的运行状况并对其进行分类，以及您的应用程序是否实现了其长期性能目标。
- 使用 Application Signals 创建并监控服务级别目标（SLO）。轻松创建并跟踪与 CloudWatch 指标相关的 SLO 的状态，包括 Application Signals 收集的新标准应用程序指标。在服务列表和拓扑图中查看并跟踪应用程序服务的服务级别指标（SLI）状态。创建警报以跟踪您的 SLO，并跟踪 Application Signals 收集的新标准应用程序指标。
- 查看 Application Signals 自动发现的应用程序拓扑图，该图可以直观地呈现您的应用程序、依赖项及其连接。
- Application Signals 可与 CloudWatch RUM、CloudWatch Synthetics 金丝雀、AWS Service Catalog AppRegistry 和 Amazon EC2 Auto Scaling 结合使用，在控制面板和地图中显示您的客户端页面、Synthetics 金丝雀和应用程序名称。

使用 Application Signals 进行日常应用程序监控

在 CloudWatch 控制台中使用 Application Signals，作为日常应用程序监控的一部分：

1. 如果您已经为服务创建了服务级别目标（SLO），请从服务级别目标（SLO）页面开始。这样，您便可即时查看最关键的服务和操作的运行状况。选择 SLO 的服务或操作名称以打开服务详细信息页面，并在排查问题时查看详细的服务信息。
2. 打开服务页面以查看所有服务的摘要，并快速查看故障率或延迟最高的服务。如果您创建了 SLO，请查看“服务”表格以了解哪些服务具有运行不正常的服务级别指标（SLI）。如果特定服务处于不正常状态，请选择该服务以打开服务详细信息页面，查看服务操作、依赖项、Synthetics Canary 和客户端请求。在图表中选择一个点以查看相关轨迹，以进行问题排查并确定操作问题的根本原因。
3. 如果已部署新服务或依赖项已更改，请打开服务地图以检查您的应用程序拓扑。查看您的应用程序地图，其中显示了客户端、Synthetics Canary、服务和依赖项之间的关系。快速查看 SLI 运行状况，查看调用量、故障率和延迟等关键指标，并深入查看服务详细信息页面中的更多详细信息。

使用 Application Signals 会产生费用。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> 无需启用 Application Signals 即可使用 CloudWatch Synthetics、CloudWatch RUM 或 CloudWatch Evidently。但是，当您将这些功能一起使用时，Synthetics 和 CloudWatch RUM 与 Application Signals 结合使用可以带来优势。

支持的语言和架构

Application Signals 支持 Java 应用程序和 Python 应用程序。

Application Signals 在 Amazon EKS、Amazon ECS 和 Amazon EC2 上受到支持并经过测试。在 Amazon EKS 集群上，Application Signals 会自动发现您的服务和集群的名称。在其他架构上，当您为 Application Signals 启用这些服务时，必须提供服务和环境的名称。

在 Amazon EC2 上启用 Application Signals 的指令应适用于任何支持 CloudWatch 代理和 AWS Distro for OpenTelemetry 的架构。但是，除了 Amazon ECS 和 Amazon EC2 之外，这些指令尚未在其他架构上进行过测试。

支持的区域

除了加拿大西部（卡尔加里）以外的所有商业区域都支持 Application Signals。

主题

- Application Signals 所需权限

- 启用 Application Signals
- 使用 Application Signals 监控应用程序的运行状况
- 收集的标准应用程序指标

Application Signals 所需权限

本部分介绍启用、管理和操作 Application Signals 所需的权限。

启用和管理 Application Signals 的权限

要管理 Application Signals，您必须使用以下权限登录：

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
        "Effect": "Allow",
        "Action": "application-signals:*",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:DescribeAlarms"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
```

```
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
        "Effect": "Allow",
        "Action": [
            "synthetics:DescribeCanaries",
            "synthetics:DescribeCanariesLastRun",
            "synthetics:GetCanaryRuns"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsRumPermissions",
        "Effect": "Allow",
        "Action": [
            "rum:BatchCreateRumMetricDefinitions",
            "rum:BatchDeleteRumMetricDefinitions",
            "rum:BatchGetRumMetricDefinitions",
            "rum:GetAppMonitor",
            "rum:GetAppMonitorData",
            "rum:ListAppMonitors",
            "rum:PutRumMetricsDestination",
            "rum:UpdateRumMetricDefinition"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsXrayPermissions",
        "Effect": "Allow",
        "Action": [
            "xray:GetTraceSummaries"
```

```
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:PutMetricAlarm",
        "Resource": [
            "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
        ]
    },
    {
        "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
            }
        }
    },
    {
        "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
        "Effect": "Allow",
        "Action": "iam:GetRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:Subscribe"
        ],
        "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
        "Effect": "Allow",
```

```
        "Action": "sns:ListTopics",
        "Resource": "*"
    }
 ]
}
```

要在 Amazon EC2、Kubernetes 或自定义架构上启用 Application Signals，请参阅[使用自定义设置在其他平台上启用 Application Signals](#)。要使用 [Amazon CloudWatch 可观测性 EKS 插件](#)在 Amazon EKS 上启用和管理 Application Signals，您需要以下权限。

> **Important**
>
> 这些权限包括带有 `Resource "*"` 的 `iam:PassRole` 与带有 `Resource "*"` 的 `eks:CreateAddon`。权限较高，应谨慎授予。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Sid": "CloudWatchApplicationSignalsEksAddonManagementPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:AccessKubernetesApi",
                "eks:CreateAddon",
                "eks:DescribeAddon",
                "eks:DescribeAddonConfiguration",
                "eks:DescribeAddonVersions",
                "eks:DescribeCluster",
                "eks:DescribeUpdate",
                "eks:ListAddons",
                "eks:ListClusters",
                "eks:ListUpdates",
                "iam:ListRoles",
                "iam:PassRole"
            ],
            "Resource": "*"
        },
        {
"Sid":
 "CloudWatchApplicationSignalsEksCloudWatchObservabilityAddonManagementPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "eks:DeleteAddon",
                "eks:UpdateAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

Application Signals 控制面板显示与您的 SLO 关联的 AWS Service Catalog AppRegistry 应用程序。要在 SLO 页面中查看这些应用程序，您必须拥有以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

操作 Application Signals

使用 Application Signals 监控服务和 SLO 的服务运营商，必须以具有以下只读权限的用户或角色身份登录账户：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
        },
        {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery",
            "logs:DescribeMetricFilters"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:GetQueryResults",
            "logs:StopQuery"
        ],
        "Resource": "*"
    },
        {
            "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:DescribeCanariesLastRun",
                "synthetics:GetCanaryRuns"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
            "Effect": "Allow",
            "Action": [
                "rum:BatchGetRumMetricDefinitions",
                "rum:GetAppMonitor",
                "rum:GetAppMonitorData",
                "rum:ListAppMonitors"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
            "Effect": "Allow",
            "Action": [
                "xray:GetTraceSummaries"
            ],
            "Resource": "*"
        }
    ]
}
```

要在 Application Signals 控制面板内，查看您的 SLO 与哪些 AWS Service Catalog AppRegistry 应用程序关联，您还需要以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
```

```
}
```

要检查是否已使用 Amazon CloudWatch 可观测性 EKS 插件在 Amazon EKS 上启用了 Application Signals，您需要拥有以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsEksReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:ListAddons",
                "eks:ListClusters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsEksDescribeAddonReadPermissions",
            "Effect": "Allow",
            "Action": [
                "eks:DescribeAddon"
            ],
            "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
        }
    ]
}
```

启用 Application Signals

本节中的主题介绍了如何在环境中启用 CloudWatch Application Signals。Amazon EKS 集群支持 Application Signals，并使用控制台设置工作流。其他平台（包括 Amazon EC2）也支持该功能，并采用自定义设置流程。

主题

- Application Signals 支持的系统
- OpenTelemetry 兼容性注意事项
- 在 Amazon EKS 集群上启用 Application Signals
- 在 Amazon EC2、Amazon ECS 或 Kubernetes 上启用 Application Signals
- 更新至所需版本的代理或 Amazon EKS 加载项

- 对 Application Signals 安装进行问题排查
- 配置 Application Signals

Application Signals 支持的系统

Application Signals 在 Amazon EKS、本机 Kubernetes、Amazon ECS 和 Amazon EC2 上受支持并经过测试。在 Amazon EC2 上启用 Application Signals 的指令应适用于任何支持 CloudWatch 代理和 AWS Distro for OpenTelemetry 的平台，但这些指令尚未在其他平台上进行过测试。

Java 兼容性

Application Signals 支持 Java 应用程序，也支持与适用于 OpenTelemetry 的 AWS Distro 相同的 Java 库和框架。有关更多信息，请参阅 Supported libraries, frameworks, application servers, and JVMs。

支持 JVM 版本 8、11 和 17。

Python 兼容性

Python 兼容性

Application Signals 支持与 AWS Distro for OpenTelemetry 相同的库和框架。有关更多信息，请参阅 opentelemetry-python-contrib 上的 Supported packages。

支持 Python 版本 3.8 及更高版本。

在为 Python 应用程序启用 Application Signals 之前，请注意以下注意事项。

- 在某些容器化应用程序中，缺少 PYTHONPATH 环境变量有时可能会导致应用程序无法启动。要解决此问题，请确保将 PYTHONPATH 环境变量设置为应用程序工作目录的位置。这是由于 OpenTelemetry 自动检测的已知问题造成的。有关此问题的更多信息，请参阅 Python autoinstrumentation setting of PYTHONPATH is not compliant（PYTHONPATH 的 Python 自动检测设置不兼容）。
- 对于 Django 应用程序，还有其他必需的配置，这些配置在 OpenTelemetry Python 文档中进行了概述。
  - 使用 `--noreload` 标志可防止自动重新加载。
  - 将 `DJANGO_SETTINGS_MODULE` 环境变量设置为 Django 应用程序 `settings.py` 文件的位置。这样可确保 OpenTelemetry 能够正确访问您的 Django 设置，并与之集成。

.NET 兼容性（预览版）

Application Signals 支持在 Amazon EKS、Amazon EC2、Amazon ECS 以及在 Amazon EC2 上运行的 Kubernetes 上将 .NET 应用程序与适用于 Open Telemetry 的 AWS Distro（ADOT）仪表化工具结合使用。

此预览版支持 .NET 8。

Application Signals 支持在 x86-64 CPU 上运行的.NET 应用程序，并且支持 Linux x64、Microsoft Windows Server 2022 x64 和 Microsoft Windows Server 2019 x64 操作系统。

OpenTelemetry 兼容性注意事项

要使用 CloudWatch Application Signals 载入您的应用程序，我们建议您事先从应用程序中完全移除所有现有的应用程序性能监控解决方案。这包括移除所有检测代码和配置。

尽管 Application Signals 使用 OpenTelemetry 检测，但不能保证与您现有的 OpenTelemetry 检测或配置兼容。在最理想的情况下，您可以保留一些 OpenTelemetry 功能，例如自定义指标。但是，请务必阅读以下部分以了解详细信息。

您已经在使用 OpenTelemetry 时的注意事项

如果您已经在应用程序中使用 OpenTelemetry，则本节的其余内容将包含实现与 Application Signals 兼容的重要信息。

- 在为应用程序启用 Application Signals 之前，必须从应用程序中移除基于 OpenTelemetry 的任何其他自动检测代理的注入。这有助于避免配置冲突。您可以继续使用兼容的 OpenTelemetry API 及 Application Signals 手动检测。
- 如果您使用手动检测从应用程序生成自定义跨度或指标，则根据检测的复杂性，启用 Application Signals 可能会导致应用程序停止生成数据或出现其他不良行为。您可以使用 OpenTelemetry 中的一些可用配置（本节后面的表格中提到的配置除外）来保留现有指标或跨度的预期行为。有关这些配置的更多信息，请参阅 OpenTelemetry 文档中的 SDK 配置。

  例如，通过使用 `OTEL_EXPORTER_OTLP_METRIC_ENDPOINT` 配置和自行管理的 OpenTelemetry Collector 实例，您可以继续将自定义指标发送到所需的目标。
- 某些环境变量或系统属性不得与 Application Signals 一起使用，但可以使用其他变量或系统属性，前提是遵循表中的指导。有关详细信息，请参阅下表。

环境变量	与 Application Signals 一起使用的建议
通用环境变量	
OTEL_SDK_DISABLED	不得设置为 true。
OTEL_TRACES_EXPORTER	必须设置为 otlp。
OTEL_LOGS_EXPORTER	设置为 none 以禁用其他日志导出程序。
OTEL_EXPORTER_OTLP_ENDPOINT	不得使用。
OTEL_EXPORTER_OTLP_METRIC_ENDPOINT	通过使用 OTEL_EXPORTER_OTLP_METRIC_ENDPOINT 配置和自行管理的 OpenTelemetry Collector 实例，您可以继续将自定义指标发送到所需的目标。 在 .NET 设置中，此环境变量支持手动仪表化，但不支持自动仪表化。您不能使用此环境变量将指标发送到自己的端点。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	不得使用。
OTEL_ATTRIBUTE_COUNT_LIMIT	如果已设置，则必须设置得足够高，以包含由 CloudWatch Application Signals 添加的约 10 个额外跨度属性。
OTEL_EXPORTER_OTLP_PROTOCOL	必须设置为 http/protobuf
OTEL_PROPAGATORS	如果已设置，则必须包含用于跟踪的 xray。
OTEL_TRACES_SAMPLER	如果已设置，则必须为 xray 以使用 X-Ray 集中采样。 要使用本地采样，请将其设置为 parentbased_traceidratio 并在 OTEL_TRACES_SAMPLER_ARG 中指定采样率。

环境变量	与 Application Signals 一起使用的建议
OTEL_TRACES_SAMPLER_ARG	如果您使用默认的 X-Ray 集中跟踪样本，则不得使用此变量。 如果您改用本地采样，请在此变量中设置采样率。例如，对于 5% 的采样率设置为 0.05。
Java 特定的环境变量	
OTEL_JAVA_ENABLED_RESOURCE_PROVIDERS	如果已设置，则必须包含 AWS 资源检测器。
Python 特定的环境变量	
OTEL_PYTHON_CONFIGURATOR	如已使用，则必须设置为 aws_configurator
OTEL_PYTHON_DISTRO	如已使用，则必须设置为 aws_distro

在 Amazon EKS 集群上启用 Application Signals

在 Amazon EKS 集群中运行的 Java 和 Python 应用程序支持 CloudWatch Application Signals。要为 Amazon EKS 集群中的应用程序启用 Application Signals，您有两种选择：

- 要在现有 Amazon EKS 集群上为应用程序启用 Application Signals，请使用 使用您的服务在 Amazon EKS 集群上启用 Application Signals 中的步骤。
- 要使用示例应用程序在非生产环境中试用 Application Signals，请使用 使用示例应用程序在新的 Amazon EKS 集群上启用 Application Signals 中的说明。此工作流使用 AWS 提供的脚本创建新的 Amazon EKS 集群并安装为 Application Signals 启用的示例应用程序。这样，您便可查看并测试 Application Signals 的端到端功能。

主题

- 使用您的服务在 Amazon EKS 集群上启用 Application Signals
- 使用示例应用程序在新的 Amazon EKS 集群上启用 Application Signals

使用您的服务在 Amazon EKS 集群上启用 Application Signals

要在现有 Amazon EKS 集群的应用程序上启用 CloudWatch Application Signals，请按照本节中的说明进行操作。

> **Important**
>
> 如果您已经在打算为 Application Signals 启用的应用程序中使用 OpenTelemetry，请在启用 Application Signals 之前参阅 OpenTelemetry 兼容性注意事项。

为现有 Amazon EKS 集群上的应用程序启用 Application Signals

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务。
3. 如果您尚未在此账户中启用 Application Signals，则必须向 Application Signals 授予发现您的服务所需的权限。为此，请执行以下操作：您的账户只需执行一次该操作。
   a. 选择开始发现您的服务。
   b. 选中该复选框并选择开始发现服务。

      首次在您的账户中完成此步骤会创建 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色。此角色授予 Application Signals 以下权限：

      - `xray:GetServiceGraph`
      - `logs:StartQuery`
      - `logs:GetQueryResults`
      - `cloudwatch:GetMetricData`
      - `cloudwatch:ListMetrics`
      - `tag:GetResources`

      有关该角色的更多信息，请参阅 CloudWatch Application Signals 的服务相关角色权限。
4. 选择启用 Application Signals。
5. 在指定平台中，选择 EKS。
6. 在选择 EKS 集群中，选择要在其中启用 Application Signals 的集群。

7. 如果此集群尚未启用 Amazon CloudWatch Observability EKS 附加组件，则系统会提示您将其启用。在这种情况下，请执行以下操作：

   a. 选择添加 CloudWatch Observability EKS 附加组件。Amazon EKS 控制台出现。

   b. 选中 Amazon CloudWatch Observability 对应的复选框，然后选择下一步。

      CloudWatch Observability EKS 附加组件启用 Application Signals 和 CloudWatch Container Insights，从而增强 Amazon EKS 的可观测性。有关安装 Container Insights 的更多信息，请参阅 Container Insights。

   c. 选择要安装的最新版本的附加组件。

   d. 选择要用于附加组件的 IAM 角色。如果您选择从节点继承，请将正确的权限附加到您的 Worker 节点使用的 IAM 角色。将 *my-worker-node-role* 替换为您的 Kubernetes Worker 节点使用的 IAM 角色。

      ```
      aws iam attach-role-policy \
      --role-name my-worker-node-role \
      --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
      --policy-arn arn:aws:iam::aws:policy/AWSXRayWriteOnlyAccess
      ```

   e. 如果您想要创建服务角色以使用附加组件，请参阅 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理。

   f. 选择下一步，确认屏幕上的信息，然后选择创建。

   g. 在下一个屏幕中，选择启用 CloudWatch Application Signals 以返回 CloudWatch 控制台并完成该过程。

8. 有两个选项可以为您的应用程序启用 Application Signals。为保持一致性，建议您为每个集群选择一个选项。

   - 控制台选项更简单。使用此方法会导致您的容器组（pod）立即重启。
   - 注释清单文件方法可以让您更好地控制容器组（pod）何时重启；如果您不想集中监控，还可以帮助您以更分散的方式管理监控。

   Console

      控制台选项使用 Amazon CloudWatch Observability EKS 附加组件的高级配置，为您的服务设置 Application Signals。有关附加组件的更多信息，请参阅（可选）其他配置。

如果没有看到工作负载和命名空间列表，则请确保您拥有查看该集群工作负载和命名空间的适当权限。有关更多信息，请参阅所需权限。

您可以监控单个工作负载或整个命名空间。

要监控单个工作负载，请执行以下操作：

1. 选中要监控的工作负载旁的复选框。
2. 使用选择语言下拉列表选择工作负载的语言。选择要为其启用 Application Signals 的语言，然后选择复选标记图标（✓）以保存此选择。

   对于 Python 应用程序，请确保您的应用程序符合所需的先决条件，然后再继续。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。
3. 选择完成。Amazon CloudWatch Observability EKS 附加组件会立即将 AWS Distro for OpenTelemetry 自动检测（ADOT）SDK 注入到您的容器组（pod）中，并触发容器组（pod）重启以允许收集应用程序指标和跟踪。

要监控整个命名空间，请执行以下操作：

1. 选中要监控的命名空间旁的复选框。
2. 使用选择语言下拉列表选择命名空间的语言。选择要为其启用 Application Signals 的语言，然后选择复选标记图标（✓）以保存此选择。此操作将该语言应用到此命名空间中的所有工作负载，无论其当前已部署还是将在未来部署。

   对于 Python 应用程序，请确保您的应用程序符合所需的先决条件，然后再继续。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。
3. 选择完成。Amazon CloudWatch Observability EKS 附加组件会立即将 AWS Distro for OpenTelemetry 自动检测（ADOT）SDK 注入到您的容器组（pod）中，并触发容器组（pod）重启以允许收集应用程序指标和跟踪。

要在另一个 Amazon EKS 集群中启用 Application Signals，请从服务屏幕中选择启用 Application Signals。

Annotate manifest file

在 CloudWatch 控制台中，监控服务部分说明了您必须向集群中的清单 YAML 添加注释。添加此注释会自动检测应用程序以向 Application Signals 发送指标、跟踪和日志。

您有两种注释选项：

- 注释工作负载会自动检测集群中的单个工作负载。
- 注释命名空间会自动检测所选命名空间中部署的所有工作负载。

选择其中一个选项，然后按照相应的步骤操作：

- 要为单个工作负载添加注释，请执行以下操作：
  1. 选择注释工作负载。
  2. 将以下行之一粘贴到工作负载清单文件的 PodTemplate 部分。
     - 对于 Java 工作负载：`annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - 对于 Python 工作负载：`annotations: instrumentation.opentelemetry.io/inject-python: "true"`

       对于 Python 应用程序，还需要额外的配置。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。
     - 对于 .NET 工作负载（预览版）：`annotations: instrumentation.opentelemetry.io/inject-dotnet: "true"`

       > **Note**
       >
       > 要为 Windows Server 实例上的 .NET 工作负载启用 Application Signals，还必须通过打开防火墙上的 TCP/2000 和 TCP/4316 端口来为 CloudWatch 代理启用网络访问权限。
  3. 在您的终端中，输入 `kubectl apply -f` *`your_deployment_yaml`* 以应用更改。
- 要为命名空间中的所有工作负载添加注释，请执行以下操作：
  1. 选择注释命名空间。
  2. 将以下行之一粘贴到命名空间清单文件的元数据部分。如果命名空间同时包含 Java 和 Python 工作负载，则请将这两行都粘贴到命名空间清单文件中。
     - 如果命名空间中有 Java 工作负载：`annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - 如果命名空间中有 Python 工作负载：`annotations: instrumentation.opentelemetry.io/inject-python: "true"`

对于 Python 应用程序，还需要额外的配置。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。

3. 在您的终端中，输入 `kubectl apply -f your_namespace_yaml` 以应用更改。
4. 在您的终端中，输入命令以重启命名空间中的所有容器组（pod）。重启部署工作负载的命令示例是 `kubectl rollout restart deployment -n namespace_name`

9. 选择完成后查看服务。这将带您进入 Application Signals 服务视图，您可以在其中查看 Application Signals 正在收集的数据。可能需要几分钟才会显示数据。

   要在另一个 Amazon EKS 集群中启用 Application Signals，请从服务屏幕中选择启用 Application Signals。

   有关服务视图的更多信息，请参阅 使用 Application Signals 监控应用程序的运行状况。

> **Note**
>
> 我们已经明确在为 Application Signals 启用 Python 应用程序时应注意的一些注意事项。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。

使用示例应用程序在新的 Amazon EKS 集群上启用 Application Signals

要在使用示例应用程序检测自己的应用程序之前，在示例应用程序上试用 CloudWatch Application Signals，请按照本节中的说明进行操作。这些说明使用脚本帮助您创建 Amazon EKS 集群、安装示例应用程序以及检测示例应用程序以使用 Application Signals。

示例应用程序是一个 Spring“Pet Clinic”应用程序，由四个微服务组成。这些服务在 Amazon EC2 上的 Amazon EKS 上运行，并利用 Application Signals 启用脚本，通过 Java 或 Python 自动检测代理来启用集群。

要求

- 目前 Application Signals 仅监控 Java 和 Python 应用程序。
- 必须已在实例上安装 AWS CLI。我们推荐 AWS CLI 版本 2，但版本 1 应该也可以使用。有关安装 AWS CLI 的更多信息，请参阅安装或更新 AWS CLI 的最新版本。
- 本节中的脚本旨在 Linux 和 macOS 环境中运行。对于 Windows 实例，我们建议您使用 AWS Cloud9 环境来运行这些脚本。有关 AWS Cloud9 的更多信息，请参阅什么是 AWS Cloud9？

- 安装支持的 kubectl 版本。您使用的 kubectl 版本与您的 Amazon EKS 集群控制面板之间必须最多只有一个次要版本的差异。例如，1.26 kubectl 客户端应使用 Kubernetes 1.25、1.26 和 1.27 集群。如果您已有 Amazon EKS 集群，可能需要配置 kubectl 的 AWS 凭证。有关更多信息，请参阅为 Amazon EKS 集群创建或更新 kubeconfig 文件。
- 安装 eksctl。eksctl 使用 AWS CLI 与 AWS 进行交互，这意味着该命令使用与 AWS CLI 相同的 AWS 凭证。有关更多信息，请参阅安装或更新 eksctl。
- 安装 jq。jq 是运行 Application Signals 启用脚本所必需的。有关更多信息，请参阅 Download jq。

步骤 1：下载脚本

要下载脚本，以使用示例应用程序设置 CloudWatch Application Signals，您可以将压缩的 GitHub 项目文件下载并解压缩到本地驱动器，也可以克隆 GitHub 项目。

要克隆项目，请打开终端窗口，并在给定工作目录中输入以下 Git 命令。

```
git clone https://github.com/aws-observability/application-signals-demo.git
```

步骤 2：构建并部署示例应用程序

要构建和推送示例应用程序映像，请按照以下说明进行操作。

步骤 3：部署并启用 Application Signals 和示例应用程序

在完成以下步骤之前，请确保您已完成 使用示例应用程序在新的 Amazon EKS 集群上启用 Application Signals 中列出的要求。

部署并启用 Application Signals 和示例应用程序

1. 输入以下 命令。将 *new-cluster-name* 替换为要用于新集群的名称。将 *region-name* 替换为 AWS 区域的名称，例如 us-west-1。

   此命令设置在启用 Application Signals 的新 Amazon EKS 集群中运行的示例应用程序。

   ```
   # this script sets up a new cluster, enables Application Signals, and deploys the
   # sample application
   cd application-signals-demo/scripts/eks/appsignals/one-step && ./setup.sh new-
   cluster-name region-name
   ```

   安装脚本大约需要 30 分钟才能运行，并执行以下操作：

- 在指定区域创建新的 Amazon EKS 集群。
- 为 Application Signals（`arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess` 和 `arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy`）创建必要的 IAM 权限。
- 通过安装 CloudWatch 代理并自动检测示例应用程序来获取 CloudWatch 指标和 X-Ray 跟踪，从而启用 Application Signals。
- 在同一 Amazon EKS 集群中部署 PetClinic Spring 示例应用程序。
- 创建五个 CloudWatch Synthetics Canary，分别名为 `pc-add-vist`、`pc-create-owners`、`pc-visit-pet`、`pc-visit-vet`、`pc-clinic-traffic`。这些 Canary 将以一分钟的频率运行，为示例应用程序生成合成流量，并演示 Synthetics Canary 在 Application Signals 中的显示方式。
- 为 PetClinic 应用程序创建四个服务级别目标（SLO），名称如下：
  - 搜索所有者的可用性
  - 搜索所有者的延迟
  - 注册所有者的可用性
  - 注册所有者的延迟
- 使用自定义信任策略创建所需的 IAM 角色，该策略授予 Application Signals 以下权限：
  - `cloudwatch:PutMetricData`
  - `cloudwatch:GetMetricData`
  - `xray:GetServiceGraph`
  - `logs:StartQuery`
  - `logs:GetQueryResults`

2. （可选）如果您想查看 PetClinic 示例应用程序的源代码，可以在根文件夹下找到。

```
- application-signals-demo
  - spring-petclinic-admin-server
  - spring-petclinic-api-gateway
  - spring-petclinic-config-server
  - spring-petclinic-customers-service
  - spring-petclinic-discovery-server
  - spring-petclinic-vets-service
  - spring-petclinic-visits-service
```

3. 要查看已部署的 PetClinic 示例应用程序，请运行以下命令查找 URL：

```
kubectl get ingress
```

步骤 4：监控示例应用程序

完成上一部分中创建 Amazon EKS 集群和部署示例应用程序的步骤后，您可以使用 Application Signals 监控应用程序。

> **Note**
>
> 要使 Application Signals 控制台开始填充，必须有一些流量到达示例应用程序。前面的一部分步骤创建了 CloudWatch Synthetics Canary，用于生成示例应用程序的流量。

服务运行状况监控

启用后，CloudWatch Application Signals 会自动发现并填充服务列表，无需任何其他设置。

查看已发现服务的列表并监控其运行状况

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、服务。
3. 要查看服务、其操作和依赖项，请在列表中选择其中一个服务的名称。

   这种以应用程序为中心的统一视图有助于全面了解用户如何与您的服务进行交互。这可以帮助您在出现性能异常时对问题进行分类。有关服务视图的完整详细信息，请参阅 使用 Application Signals 监控应用程序的运行状况。
4. 选择服务操作选项卡，查看该服务操作的标准应用程序指标。例如，这些操作是服务调用的 API 操作。

   然后，要查看该服务的单个操作的图表，请选择该操作的名称。
5. 选择依赖项选项卡，查看您的应用程序具有的依赖项，以及每个依赖项的关键应用程序指标。依赖项包括您的应用程序调用的 AWS 服务和第三方服务。
6. 要从服务详细信息页面查看相关跟踪，请在表格上方的三个图表中选择一个数据点。这将在新窗格中填充该时间段中经过筛选的跟踪。这些跟踪将根据您选择的图表进行排序和筛选。例如，如果您选择延迟图表，则跟踪将按服务响应时间排序。

7. 在 CloudWatch 控制台导航窗格中，选择 SLO。您将看到脚本为示例应用程序创建的 SLO。有关 SLO 的更多信息，请参阅 服务级别目标（SLO）。

（可选）第 5 步：清理

Application Signals 测试完成后，您可以使用 Amazon 提供的脚本来清理并删除您账户中为示例应用程序创建的构件。要执行清理，请输入以下命令。将 *new-cluster-name* 替换为您为示例应用程序创建的集群名称，并将 *region*-name 替换为 AWS 区域的名称，例如 us-west-1。

```
cd application-signals-demo/scripts/eks/appsignals/one-step && ./cleanup.sh new-
cluster-name region-name
```

在 Amazon EC2、Amazon ECS 或 Kubernetes 上启用 Application Signals

使用这些部分中的自定义设置步骤，在 Amazon EKS 以外的平台上启用 CloudWatch Application Signals。在这些架构上，您可以自行安装并配置 CloudWatch 代理和 AWS Distro for OpenTelemetry。

在这些架构上，Application Signals 不会自动发现您的服务或其集群或主机的名称。您必须在自定义设置期间指定这些名称，而您指定的名称就是显示在 Application Signals 控制面板上的名称。

主题

- 使用自定义设置在 Amazon ECS 上启用 Application Signals
- 在 Amazon EC2 上启用 Application Signals
- 在 Kubernetes 平台上启用 Application Signals

使用自定义设置在 Amazon ECS 上启用 Application Signals

使用这些自定义设置说明，将您在 Amazon ECS 上的应用程序载入到 CloudWatch Application Signals。您可以自行安装并配置 CloudWatch 代理和 AWS Distro for OpenTelemetry。

> **Important**
>
> 仅支持 awsvpc 网络模式。EC2 和 Fargate 启动类型均受支持。

在 Amazon ECS 集群上，Application Signals 不会自动发现您的服务名称。您必须在自定义设置期间指定服务名称，而您指定的名称就是显示在 Application Signals 控制面板上的名称。

步骤 1：在账户中启用 Application Signals

如果您尚未在此账户中启用 Application Signals，则必须向 Application Signals 授予发现您的服务所需的权限。为此，请执行以下操作：您的账户只需执行一次该操作。

为您的应用程序启用 Application Signals

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务。
3. 选择开始发现您的服务。
4. 选中该复选框并选择开始发现服务。

   首次在您的账户中完成此步骤会创建 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色。此角色授予 Application Signals 以下权限：

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`
   - `tag:GetResources`

   有关该角色的更多信息，请参阅 CloudWatch Application Signals 的服务相关角色权限。

步骤 2：创建 IAM 角色

您必须创建一个 IAM 角色。如果您已创建此角色，则可能需要向其添加权限。

- ECS 任务角色 – 容器使用此角色运行。权限应是您的应用程序所需的任何权限，以及 CloudWatchAgentServerPolicy。

有关创建 IAM 角色的更多信息，请参阅创建 IAM 角色。

步骤 3：准备 CloudWatch 代理配置

首先，在启用 Application Signals 的情况下准备代理配置。为此，请创建一个名为 `/tmp/ecs-cwagent.json` 的本地文件。

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

然后，将此配置上传到 SSM 参数仓库中。要执行此操作，请输入以下命令。在该文件中，将 *$REGION* 替换为您的实际区域名称。

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

步骤 4：使用 CloudWatch 代理检测您的应用程序

下一步是为 CloudWatch Application Signals 检测您的应用程序。

Java

使用 CloudWatch 代理检测您在 Amazon ECS 上的应用程序

1. 首先，指定绑定挂载。在接下来的步骤中，该卷将用于跨容器共享文件。您将在此过程的稍后部分使用此绑定挂载。

   ```
   "volumes": [
     {
       "name": "opentelemetry-auto-instrumentation"
     }
   ]
   ```

2. 添加 CloudWatch 代理 sidecar 定义。为此，请在应用程序的任务定义中追加一个名为 `ecs-cwagent` 的新容器。将 *$REGION* 替换为您的实际区域名称。将 *$IMAGE* 替换为 Amazon

Elastic Container Registry 上最新 CloudWatch 容器映像的路径。有关更多信息，请参阅 Amazon ECR 上的 cloudwatch-agent。

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. 在应用程序的任务定义中添加一个新容器 `init`。将 *$IMAGE* 替换为 AWS Distro for OpenTelemetry Amazon ECR 映像存储库中的最新映像。

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
```

```
}
```

4. 将以下环境变量添加到您的应用程序容器中。您必须使用适用于 OpenTelemetry 的 AWS Distro Java 自动仪表化代理版本 1.32.2 或更高版本。

环境变量	启用 Application Signals 的设置
OTEL_RESOURCE_ATTRIBUTES	将以下信息指定为键值对： • service.name 设置服务的名称。这将作为应用程序的服务名称显示在 Application Signals 控制面板中。如果您不提供此键的值，将使用默认值 UnknownService 。 • deployment.environment 设置应用程序运行所在的环境。这将作为应用程序的托管环境显示在 Application Signals 控制面板中。如果您未指定此项，则使用默认值 generic:default 。 此属性键仅供 Application Signals 使用，并转换为 X-Ray 跟踪注释和 CloudWatch 指标维度。 （可选）要启用 Application Signals 的日志关联，请将其他环境变量 aws.log.group.names 设置为应用程序日志的日志组名称。这样，应用程序的跟踪和指标便可与该特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：aws.log.group.names=log-group-1&log-group-2 。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日

环境变量	启用 Application Signals 的设置
	志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。
OTEL_AWS_APPLICATION_SIGNALS_ENABLED	设置为 true 可使您的容器开始向 Application Signals 发送 X-Ray 跟踪和 CloudWatch 指标。
OTEL_METRICS_EXPORTER	设置为 none 可禁用其他指标导出程序。
OTEL_LOGS_EXPORTER	设置为 none 以禁用其他日志导出程序。
OTEL_EXPORTER_OTLP_PROTOCOL	设置为 http/protobuf 以使用 HTTP 将指标和跟踪发送到 Application Signals。
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	设置为 http://localhost:4316/v1/metrics 可将指标发送到 CloudWatch sidecar。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	设置为 http://localhost:4316/v1/traces 可向 CloudWatch sidecar 发送跟踪。
OTEL_TRACES_SAMPLER	将此项设置为 xray 以将 X-Ray 设置为跟踪取样器。
OTEL_PROPAGATORS	将 xray 设置为传播器之一。
JAVA_TOOL_OPTIONS	设置为 " -javaagent:$ *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* "。将 *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* 替换为存储 AWS Distro for OpenTelemetry Java 自动检测代理的路径。例如，/otel-auto-instrumentation/javaagent.jar

5. 挂载您在此过程的步骤 1 中定义的卷 opentelemetry-auto-instrumentation。如果您不需要启用与指标和跟踪的日志关联，请使用以下 Java 应用程序示例。如果要启用日志关联，请改为参阅下一步。

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
```

```
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

6. （可选）要启用日志关联，请在挂载卷之前执行以下操作。在OTEL_RESOURCE_ATTRIBUTES中，为应用程序的日志组设置其他环境变量 aws.log.group.names。这样，应用程序的跟踪和指标便可与这些特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：aws.log.group.names=log-group-1&log-group-2。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。

   示例如下：使用此示例可在您挂载于此过程的步骤 1 中定义的卷 opentelemetry-auto-instrumentation 时启用日志关联。

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
```

```
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

在为 Python 应用程序启用 Application Signals 之前，请注意以下注意事项。

- 在某些容器化应用程序中，缺少 PYTHONPATH 环境变量有时可能会导致应用程序无法启动。要解决此问题，请确保将 PYTHONPATH 环境变量设置为应用程序工作目录的位置。这是由于 OpenTelemetry 自动检测的已知问题造成的。有关此问题的更多信息，请参阅 Python autoinstrumentation setting of PYTHONPATH is not compliant（PYTHONPATH 的 Python 自动检测设置不兼容）。
- 对于 Django 应用程序，还有其他必需的配置，这些配置在 OpenTelemetry Python 文档中进行了概述。
  - 使用 `--noreload` 标志可防止自动重新加载。
  - 将 `DJANGO_SETTINGS_MODULE` 环境变量设置为 Django 应用程序 `settings.py` 文件的位置。这样可确保 OpenTelemetry 能够正确访问您的 Django 设置，并与之集成。

使用 CloudWatch 代理检测您在 Amazon ECS 上的 Python 应用程序

1. 首先，指定绑定挂载。在接下来的步骤中，该卷将用于跨容器共享文件。您将在此过程的稍后部分使用此绑定挂载。

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. 添加 CloudWatch 代理 sidecar 定义。为此，请在应用程序的任务定义中追加一个名为 `ecs-cwagent` 的新容器。将 *$REGION* 替换为您的实际区域名称。将 *$IMAGE* 替换为 Amazon Elastic Container Registry 上最新 CloudWatch 容器映像的路径。有关更多信息，请参阅 Amazon ECR 上的 cloudwatch-agent。

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
```

```
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. 在应用程序的任务定义中添加一个新容器 `init`。将 *$IMAGE* 替换为 AWS Distro for OpenTelemetry Amazon ECR 映像存储库中的最新映像。

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
}
```

4. 将以下环境变量添加到您的应用程序容器中。

环境变量	启用 Application Signals 的设置
`OTEL_RESOURCE_ATTRIBUTES`	将以下信息指定为键值对： • `service.name` 设置服务的名称。这将作为应用程序的服务名称显示在 Application Signals 控制面板中。如果您不提供此键的值，将使用默认值 `UnknownService`。

环境变量	启用 Application Signals 的设置
	• `deployment.environment` 设置应用程序运行所在的环境。这将作为应用程序的托管环境显示在 Application Signals 控制面板中。如果您未指定此项，则使用默认值 `generic:default` 。 此属性键仅供 Application Signals 使用，并转换为 X-Ray 跟踪注释和 CloudWatch 指标维度。 （可选）要启用 Application Signals 的日志关联，请将其他环境变量 `aws.log.group.names` 设置为应用程序日志的日志组名称。这样，应用程序的跟踪和指标便可与该特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：`aws.log.group.names=log-group-1&log-group-2` 。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	设置为 `true` 可使您的容器开始向 Application Signals 发送 X-Ray 跟踪和 CloudWatch 指标。
`OTEL_METRICS_EXPORTER`	设置为 `none` 可禁用其他指标导出程序。
`OTEL_EXPORTER_OTLP_PROTOCOL`	设置为 `http/protobuf` 以使用 HTTP 向 CloudWatch 发送指标和跟踪。

环境变量	启用 Application Signals 的设置
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	设置为 http://127.0.0.1:4316/v1/metrics 可将指标发送到 CloudWatch sidecar。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	设置为 http://127.0.0.1:4316/v1/traces 可向 CloudWatch sidecar 发送跟踪。
OTEL_TRACES_SAMPLER	将此项设置为 xray 以将 X-Ray 设置为跟踪取样器。
OTEL_PROPAGATORS	将 xray 添加为传播器之一。
OTEL_PYTHON_DISTRO	设置为 aws_distro 以使用 ADOT Python 工具。
OTEL_PYTHON_CONFIGURATOR	设置为 aws_configuration 以使用 ADOT Python 配置。
PYTHONPATH	将 $APP_PATH 替换为容器中应用程序工作目录的位置。Python 解释器查找您的应用程序模块需要执行此操作。
DJANGO_SETTINGS_MODULE	仅 Django 应用程序需要。将其设置为 Django 应用程序 settings.py 文件的位置。替换 $PATH_TO_SETTINGS 。

5. 挂载您在此过程的步骤 1 中定义的卷 opentelemetry-auto-instrumentation-python。如果您不需要启用与指标和跟踪的日志关联，请使用以下 Python 应用程序示例。如果要启用日志关联，请改为参阅下一步。

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
```

```
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
```

```
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

6. （可选）要启用日志关联，请在挂载卷之前执行以下操作。在OTEL_RESOURCE_ATTRIBUTES中，为应用程序的日志组设置其他环境变量 aws.log.group.names。这样，应用程序的跟踪和指标便可与这些特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：aws.log.group.names=log-group-1&log-group-2。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。

   示例如下：要启用日志关联，请在挂载您于此过程的步骤 1 中定义的卷 opentelemetry-auto-instrumentation-python 时使用此示例。

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
```

```
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
"aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
```

```
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

.NET (Preview)

使用 CloudWatch 代理检测您在 Amazon ECS 上的应用程序

1. 首先，指定绑定挂载。在接下来的步骤中，该卷将用于跨容器共享文件。您将在此过程的稍后部分使用此绑定挂载。

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. 添加 CloudWatch 代理 sidecar 定义。为此，请在应用程序的任务定义中追加一个名为 `ecs-cwagent` 的新容器。将 *$REGION* 替换为您的实际区域名称。将 *$IMAGE* 替换为 Amazon Elastic Container Registry 上最新 CloudWatch 容器映像的路径。有关更多信息，请参阅 Amazon ECR 上的 cloudwatch-agent。

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
```

```
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. 在应用程序的任务定义中添加一个新容器 `init`。将 *$IMAGE* 替换为 AWS Distro for OpenTelemetry Amazon ECR 映像存储库中的最新映像。

   对于 Linux 容器实例，请使用以下代码。

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "cp",
      "-a",
      "autoinstrumentation/.",
      "/otel-auto-instrumentation"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "/otel-auto-instrumentation",
          "readOnly": false
      }
  ]
}
```

   对于 Windows Server 容器实例，请使用以下代码。

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "CMD",
      "/c",
      "xcopy",
```

```
        "/e",
        "C:\\autoinstrumentation\\*",
        "C:\\otel-auto-instrumentation",
        "&&",
        "icacls",
        "C:\\otel-auto-instrumentation",
        "/grant",
        "*S-1-1-0:R",
        "/T"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ]
}
```

4. 在 init 容器上添加依赖项，以确保容器在应用程序容器启动之前完成。

```
"dependsOn": [
    {
        "containerName": "init",
        "condition": "SUCCESS"
    }
]
```

5. 将以下环境变量添加到您的应用程序容器中。您必须使用适用于 OpenTelemetry 的 AWS Distro .NET 自动仪表化代理版本 1.1.0 或更高版本。

环境变量	启用 Application Signals 的设置
OTEL_RESOURCE_ATTRIBUTES	将以下信息指定为键值对： • service.name 设置服务的名称。这将作为应用程序的服务名称显示在 Application Signals 控制面板中。如果您不提供此键的值，将使用默认值 UnknownService 。

环境变量	启用 Application Signals 的设置
	• deployment.environment 设置应用程序运行所在的环境。这将作为应用程序的托管环境显示在 Application Signals 控制面板中。如果您未指定此项，则使用默认值 generic:default 。 此属性键仅供 Application Signals 使用，并转换为 X-Ray 跟踪注释和 CloudWatch 指标维度。
OTEL_AWS_APPLICATION_SIGNALS_ENABLED	设置为 true 可使您的容器开始向 Application Signals 发送 X-Ray 跟踪和 CloudWatch 指标。
OTEL_METRICS_EXPORTER	设置为 none 可禁用其他指标导出程序。
OTEL_LOGS_EXPORTER	设置为 none 以禁用其他日志导出程序。
OTEL_EXPORTER_OTLP_PROTOCOL	设置为 http/protobuf 以使用 HTTP 将指标和跟踪发送到 Application Signals。
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	设置为 http://localhost:4316/v1/metrics 可将指标发送到 CloudWatch sidecar。
OTEL_EXPORTER_OTLP_ENDPOINT	设置为 http://localhost:4316/ 可向 CloudWatch sidecar 发送跟踪。
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	设置为 http://localhost:4316/v1/traces 可向 CloudWatch sidecar 发送跟踪。
OTEL_DOTNET_AUTO_HOME	设置为 ADOT .NET 自动仪表化工具的安装位置。

环境变量	启用 Application Signals 的设置
OTEL_DOTNET_AUTO_PLUGINS	设置为 AWS.OpenTelemetry.AutoInstrumentation.Plugin, AWS.OpenTelemetry.AutoInstrumentation 以启用 Application Signals 插件。
CORECLR_ENABLE_PROFILING	设置为 1 以启用分析器。
CORECLR_PROFILER	设置为 {918728DD-259F-4A6A-AC2B-B85E1B658318} 以作为分析器的 CLSID。
CORECLR_PROFILER_PATH	将此设置为分析器的路径。 在 Linux 上，将其设置为 ${OTEL_DOTNET_AUTO_HOME}/linux-x64/OpenTelemetry.AutoInstrumentation.Native.so 在 Windows Server 上，将其设置为 ${OTEL_DOTNET_AUTO_HOME}/win-x64/OpenTelemetry.AutoInstrumentation.Native.dll
DOTNET_ADDITIONAL_DEPS	将此设置为 ${OTEL_DOTNET_AUTO_HOME}/AdditionalDeps 的文件夹路径。
DOTNET_SHARED_STORE	将此设置为 ${OTEL_DOTNET_AUTO_HOME}/store 的文件夹路径。
DOTNET_STARTUP_HOOKS	将此设置为要在主应用程序入口点之前运行的托管程序集 ${OTEL_DOTNET_AUTO_HOME}/net/OpenTelemetry.AutoInstrumentation.StartupHook.dll 的路径。

6. 挂载您在此过程的步骤 1 中定义的卷 opentelemetry-auto-instrumentation。对于 Linux，请使用以下代码。

```
{
    "name": "my-app",
   ...
    "environment": [
        {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "/otel-auto-instrumentation/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "/otel-auto-instrumentation/AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "/otel-auto-instrumentation/store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "/otel-auto-instrumentation/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "/otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
```

```
            "value": "AWS.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=aws-otel-integ-test"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
           "name": "OTEL_TRACES_SAMPLER_ARG",
           "value": "endpoint=http://localhost:2000"
       },
```

```
        {
            "name": "OTEL_PROPAGATORS",
            "value": "tracecontext,baggage,b3,xray"
        }
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "/otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

对于 Windows Server，使用以下代码。

```
{
    "name": "my-app",
   ...
    "environment": [
       {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "C:\\otel-auto-instrumentation\\win-x64\
\OpenTelemetry.AutoInstrumentation.Native.dll"
        },
```

```
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "C:\\otel-auto-instrumentation\\AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "C:\\otel-auto-instrumentation\\store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "C:\\otel-auto-instrumentation\\net\
\OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "C:\\otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=aws-otel-integ-test"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
```

```
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
           "name": "OTEL_TRACES_SAMPLER_ARG",
           "value": "endpoint=http://localhost:2000"
       },
        {
            "name": "OTEL_PROPAGATORS",
            "value": "tracecontext,baggage,b3,xray"
        }
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

步骤 5：部署应用程序

创建任务定义的新修订版并将其部署到您的应用程序集群。您应该在新创建的任务中看到三个容器：

- `init`
- `ecs-cwagent`
- *`my-app`*

在 Amazon EC2 上启用 Application Signals

对于在 Amazon EC2 或其他实例类型上运行的应用程序，您可以自行安装并配置 CloudWatch 代理和 AWS Distro for OpenTelemetry。在这些启用了自定义 Application Signals 设置的架构上，Application Signals 不会自动发现您的服务的名称或运行这些服务的主机或集群。您必须在自定义设置期间指定这些名称，而您指定的名称就是显示在 Application Signals 控制面板上的名称。

本节中的说明适用于 Java 应用程序和 Python 应用程序。这些步骤已在 Amazon EC2 实例上进行了测试，但预计也可以在支持 AWS Distro for OpenTelemetry 的其他架构上运行。

要求

- 要获得对 Application Signals 的支持，您必须同时使用最新版本的 CloudWatch 代理和 AWS Distro for OpenTelemetry 代理。
- 必须已在实例上安装 AWS CLI。我们推荐 AWS CLI 版本 2，但版本 1 应该也可以使用。有关安装 AWS CLI 的更多信息，请参阅安装或更新 AWS CLI 的最新版本。

> **Important**
>
> 如果您已经在打算为 Application Signals 启用的应用程序中使用 OpenTelemetry，请在启用 Application Signals 之前参阅 OpenTelemetry 兼容性注意事项。

步骤 1：在账户中启用 Application Signals

如果您尚未在此账户中启用 Application Signals，则必须向 Application Signals 授予发现您的服务所需的权限。为此，请执行以下操作：您的账户只需执行一次该操作。

为您的应用程序启用 Application Signals

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务。
3. 选择开始发现您的服务。

4. 选中该复选框并选择开始发现您的服务。

   首次在您的账户中完成此步骤时，Application Signals 会创建 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色。此角色授予 Application Signals 以下权限：

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`
   - `tag:GetResources`

   有关该角色的更多信息，请参阅 CloudWatch Application Signals 的服务相关角色权限。

步骤 2：下载并启动 CloudWatch 代理

作为在 Amazon EC2 实例或本地主机上启用 Application Signals 的一部分安装 CloudWatch 代理

1. 将最新版本的 CloudWatch 代理下载到实例。如果实例已安装 CloudWatch 代理，您可能需要对其进行更新。只有在 2023 年 11 月 30 日或之后发布的代理版本支持 CloudWatch Application Signals。

   有关下载 CloudWatch 代理的信息，请参阅 下载 CloudWatch 代理软件包。

2. 启动 CloudWatch 代理之前，请将其配置为启用 Application Signals。以下示例是 CloudWatch 代理配置，该配置为 EC2 主机上的指标和跟踪启用 Application Signals。

   建议您将此文件放在 Linux 系统中的 `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`。

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
```

```
      "application_signals": {}
    }
  }
}
```

3. 将 CloudWatchAgentServerPolicy IAM 策略附加到 Amazon EC2 实例的 IAM 角色。有关本地主机的权限，请参阅 本地服务器的权限。

   a. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。

   b. 选择角色并查找您的 Amazon EC2 实例使用的角色。然后选择该角色的名称。

   c. 在权限选项卡中，选择添加权限、附加策略。

   d. 查找 CloudWatchAgentServerPolicy。如果需要，请使用搜索框。然后选择该策略对应的复选框，选择添加权限。

4. 通过输入以下命令启动 CloudWatch 代理。将 *agent-config-file-path* 替换为指向 CloudWatch 代理配置文件的路径，例如 `./amazon-cloudwatch-agent.json`。必须包含 `file:` 前缀，如图所示。

```
export CONFIG_FILE_PATH=./amazon-cloudwatch-agent.json
```

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config \
-m ec2 -s -c file:agent-config-file-path
```

本地服务器的权限

对于本地主机，您需要为设备提供 AWS 授权。

设置本地主机的权限

1. 创建用于向本地主机提供权限的 IAM 用户：

   a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。

   b. 依次选择用户、创建用户。

   c. 在用户详情中，为用户名输入新 IAM 用户的名称。这是 AWS 的登录名，将用于对您的主机进行身份验证。然后选择下一个

   d. 在设置权限页面的权限选项下，选择直接附加策略。

e. 从权限策略列表中，选择要添加到您的用户的 CloudWatchAgentServerPolicy 策略。然后选择下一步。

f. 在查看并创建页面上，确保您对用户名满意，并且 CloudWatchAgentServerPolicy 策略位于权限摘要中。

g. 选择 Create user。

2. 创建和检索您的 AWS 访问密钥和秘密密钥：

a. 在 IAM 控制台的导航窗格中，选择用户，然后选择您在上一步中所创建用户的用户名。

b. 在用户的页面上，选择安全凭证选项卡。然后，在访问密钥部分，选择创建访问密钥。

c. 对于创建访问密钥步骤 1，选择命令行界面 (CLI)。

d. 对于创建访问密钥步骤 2，（可选）输入标记，然后选择下一步。

e. 对于创建访问密钥步骤 3，选择下载.csv 文件以保存包含您的 IAM 用户访问密钥和秘密访问密钥的 .csv 文件。您在后续步骤中需要此信息。

f. 选择完成。

3. 通过输入以下命令，在本地主机中配置 AWS 凭证。将 *ACCESS_KEY_ID* 和 *SECRET_ACCESS_ID* 替换为您在上一步中下载的 .csv 文件中新生成的访问密钥和秘密访问密钥。

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

步骤 3：检测您的应用程序并将其启动

下一步是为 CloudWatch Application Signals 检测您的应用程序。

Java

检测 Java 应用程序，作为在 Amazon EC2 实例或本地主机上启用 Application Signals 的一部分流程

1. 下载最新版本的 AWS Distro for OpenTelemetry Java 自动检测代理。您可以使用此链接下载最新版本。您可以在 aws-otel-java-instrumentation 版本上查看有关所有已发布版本的信息。

2. 要优化 Application Signals 的优势，请在启动应用程序之前使用环境变量提供更多信息。此信息将显示在 Application Signals 控制面板中。

   - 对于 OTEL_RESOURCE_ATTRIBUTES 变量，将以下信息指定为键值对：

     - （可选）service.name 设置服务的名称。这将作为应用程序的服务名称在 Application Signals 控制面板中显示。如果您不提供此键的值，将使用默认值 UnknownService。
     - （可选）deployment.environment 设置应用程序运行所在的环境。这将作为应用程序的托管环境显示在 Application Signals 控制面板中。如果您未指定此项，则使用以下默认值之一：
       - 如果这是属于自动扩缩组的实例，则将其设置为 ec2:*name-of-Auto-Scaling-group*
       - 如果这是不属于自动扩缩组的 Amazon EC2 实例，则将其设置为 ec2:default
       - 如果这是本地主机，则将其设置为 generic:default

       此环境变量仅供 Application Signals 使用，并转换为 X-Ray 跟踪注释和 CloudWatch 指标维度。
   - 对于 OTEL_EXPORTER_OTLP_TRACES_ENDPOINT 变量，指定要将跟踪导出到的基本端点 URL。CloudWatch 代理将 4316 作为其 OTLP 端口公开。在 Amazon EC2 上，由于应用程序与本地 CloudWatch 代理通信，因此您应将此值设置为 OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces
   - 对于 OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT 变量，指定要将指标导出到的基本端点 URL。CloudWatch 代理将 4316 作为其 OTLP 端口公开。在 Amazon EC2 上，由于应用程序与本地 CloudWatch 代理通信，因此您应将此值设置为 OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
   - 对于 JAVA_TOOL_OPTIONS 变量，指定存储 AWS Distro for OpenTelemetry Java 自动检测代理的路径。

     ```
     export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
     ```

     例如：

     ```
     export AWS_ADOT_JAVA_INSTRUMENTATION_PATH=./aws-opentelemetry-agent.jar
     ```

- 对于 OTEL_METRICS_EXPORTER 变量，我们建议您将该值设置为 none。这将禁用其他指标导出程序，因此仅使用 Application Signals 导出程序。
- 将 OTEL_AWS_APPLICATION_SIGNALS_ENABLED 设置为 true。这会从跟踪中生成 Application Signals 指标。

3. 使用上一步中列出的环境变量启动应用程序。以下是启动脚本的示例。

> **Note**
>
> 以下配置仅支持适用于 Java 的 AWS Distro for OpenTelemetry 自动检测代理版本 1.32.2 及更高版本。

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORT=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
java -jar $MY_JAVA_APP.jar
```

4. （可选）要启用跟踪日志关联，请在 OTEL_RESOURCE_ATTRIBUTES 中为应用程序的日志组设置其他环境变量 aws.log.group.names。这样，应用程序的跟踪和指标便可与这些特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：aws.log.group.names=log-group-1&log-group-2。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。

   以下是帮助启用日志关联的启动脚本示例。

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORT=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
```

```
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_JAVA_APP.jar
```

Python

检测 Python 应用程序，作为在 Amazon EC2 实例上启用 Application Signals 的一部分流程

1. 下载最新版本的 AWS Distro for OpenTelemetry Python 自动检测代理。通过运行以下 命令安装它。

   ```
   pip install aws-opentelemetry-distro
   ```

   您可以在 AWS Distro for OpenTelemetry Python instrumentation 上查看有关所有已发布版本的信息。

2. 要优化 Application Signals 的优势，请在启动应用程序之前使用环境变量提供更多信息。此信息将显示在 Application Signals 控制面板中。

   a. 对于 `OTEL_RESOURCE_ATTRIBUTES` 变量，将以下信息指定为键值对：

   - `service.name` 设置服务的名称。这将作为应用程序的服务名称显示在 Application Signals 控制面板中。如果您不提供此键的值，将使用默认值 `UnknownService`。
   - `deployment.environment` 设置应用程序运行所在的环境。这将作为应用程序的托管环境显示在 Application Signals 控制面板中。如果您未指定此项，则使用以下默认值之一：
     - 如果这是属于自动扩缩组的实例，则将其设置为 `ec2:`*`name-of-Auto-Scaling-group`*。
     - 如果这是不属于自动扩缩组的 Amazon EC2 实例，则将其设置为 `ec2:default`
     - 如果这是本地主机，则将其设置为 `generic:default`

     此属性键仅供 Application Signals 使用，并转换为 X-Ray 跟踪注释和 CloudWatch 指标维度。

b. 对于 OTEL_EXPORTER_OTLP_PROTOCOL 变量，指定 http/protobuf 以通过 HTTP 将遥测数据导出到以下步骤中列出的 CloudWatch 代理端点。

c. 对于 OTEL_EXPORTER_OTLP_TRACES_ENDPOINT 变量，指定要将跟踪导出到的基本端点 URL。CloudWatch 代理将 4316 作为其 HTTP 上的 OTLP 端口公开。在 Amazon EC2 上，由于应用程序与本地 CloudWatch 代理通信，因此您应将此值设置为 OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces

d. 对于 OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT 变量，指定要将指标导出到的基本端点 URL。CloudWatch 代理将 4316 作为其 HTTP 上的 OTLP 端口公开。在 Amazon EC2 上，由于应用程序与本地 CloudWatch 代理通信，因此您应将此值设置为 OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics

e. 对于 OTEL_METRICS_EXPORTER 变量，我们建议您将该值设置为 none。这将禁用其他指标导出程序，因此仅使用 Application Signals 导出程序。

f. 将 OTEL_AWS_APPLICATION_SIGNALS_ENABLED 变量设置为 true 以使您的容器开始向 Application Signals 发送 X-Ray 跟踪和 CloudWatch 指标。

3. 使用上一步中讨论的环境变量启动应用程序。以下是启动脚本的示例。

- 将 $SVC_NAME 替换为您的应用程序的名称。这将作为应用程序的名称在 Application Signals 控制面板中显示。
- 将 $PYTHON_APP 替换为您的应用程序的位置和名称。

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
opentelemetry-instrument python $MY_PYTHON_APP.py
```

在为 Python 应用程序启用 Application Signals 之前，请注意以下注意事项。

- 在某些容器化应用程序中，缺少 PYTHONPATH 环境变量有时可能会导致应用程序无法启动。要解决此问题，请确保将 PYTHONPATH 环境变量设置为应用程序工作目录的位置。这是由于 OpenTelemetry 自动检测的已知问题造成的。有关此问题的更多信息，请参阅 Python autoinstrumentation setting of PYTHONPATH is not compliant（PYTHONPATH 的 Python 自动检测设置不兼容）。
- 对于 Django 应用程序，还有其他必需的配置，这些配置在 OpenTelemetry Python 文档中进行了概述。
  - 使用 `--noreload` 标志可防止自动重新加载。
  - 将 `DJANGO_SETTINGS_MODULE` 环境变量设置为 Django 应用程序 `settings.py` 文件的位置。这样可确保 OpenTelemetry 能够正确访问您的 Django 设置，并与之集成。

4. （可选）要启用跟踪日志关联，请在 `OTEL_RESOURCE_ATTRIBUTES` 中为应用程序的日志组设置其他环境变量 `aws.log.group.names`。这样，应用程序的跟踪和指标便可与这些特定日志组的相关日志条目相关联。对于此变量，将 *$YOUR_APPLICATION_LOG_GROUP* 替换为应用程序的日志组名称。如果您有多个日志组，则可以使用和号（&）分隔它们，如以下示例所示：`aws.log.group.names=log-group-1&log-group-2`。要启用指标与日志关联的功能，设置此当前环境变量已足够。有关更多信息，请参阅 启用指标与日志关联。要启用跟踪与日志关联，您还需要更改应用程序中的日志记录配置。有关更多信息，请参阅 启用跟踪与日志关联。

   以下是帮助启用日志关联的启动脚本示例。

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_PYTHON_APP.jar
```

.NET (Preview)

在 Amazon EC2 实例或本地主机上启用 Application Signals 过程中仪表化 .NET 应用程序

1. 下载最新版本适用于 OpenTelemetry 的 AWS Distro .NET 自动仪表化工具包。可以在 aws-otel-dotnet-instrumenting Release 上下载最新版本。
2. 要启用 Application Signals，请在启动应用程序之前设置以下环境变量以提供更多信息。在启动 .NET 应用程序之前，为 .NET 仪表化设置启动钩子将需要这些变量。

   - 以下是 Linux 的示例。

```
export INSTALL_DIR=OpenTelemetryDistribution
export CORECLR_ENABLE_PROFILING=1
export CORECLR_PROFILER={918728DD-259F-4A6A-AC2B-B85E1B658318}
export CORECLR_PROFILER_PATH=${INSTALL_DIR}/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so
export DOTNET_ADDITIONAL_DEPS=${INSTALL_DIR}/AdditionalDeps
export DOTNET_SHARED_STORE=${INSTALL_DIR}/store
export DOTNET_STARTUP_HOOKS=${INSTALL_DIR}/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll
export OTEL_DOTNET_AUTO_HOME=${INSTALL_DIR}

export
 OTEL_DOTNET_AUTO_PLUGINS="AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

export OTEL_RESOURCE_ATTRIBUTES=service.name=aws-otel-integ-test
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_ENDPOINT=http://127.0.0.1:4316
export OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://127.0.0.1:4316/
v1/metrics
export OTEL_METRICS_EXPORTER=none
export OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
export OTEL_TRACES_SAMPLER=xray
export OTEL_TRACES_SAMPLER_ARG=http://127.0.0.1:2000
```

   - 以下示例适用于 Windows Server。

```
$env:INSTALL_DIR = "OpenTelemetryDistribution"
$env:CORECLR_ENABLE_PROFILING = 1
$env:CORECLR_PROFILER = "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
```

```
$env:CORECLR_PROFILER_PATH = Join-Path $env:INSTALL_DIR "win-x64/
OpenTelemetry.AutoInstrumentation.Native.dll"
$env:DOTNET_ADDITIONAL_DEPS = Join-Path $env:INSTALL_DIR "AdditionalDeps"
$env:DOTNET_SHARED_STORE = Join-Path $env:INSTALL_DIR "store"
$env:DOTNET_STARTUP_HOOKS = Join-Path $env:INSTALL_DIR "net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
$env:OTEL_DOTNET_AUTO_HOME = $env:INSTALL_DIR

$env:OTEL_DOTNET_AUTO_PLUGINS =
 "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

$env:OTEL_RESOURCE_ATTRIBUTES = "service.name=aws-otel-integ-test"
$env:OTEL_EXPORTER_OTLP_PROTOCOL = "http/protobuf"
$env:OTEL_EXPORTER_OTLP_ENDPOINT = "http://127.0.0.1:4316"
$env:OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT = "http://127.0.0.1:4316/
v1/metrics"
$env:OTEL_METRICS_EXPORTER = "none"
$env:OTEL_AWS_APPLICATION_SIGNALS_ENABLED = "true"
$env:OTEL_TRACES_SAMPLER = "xray"
$env:OTEL_TRACES_SAMPLER_ARG = "http://127.0.0.1:2000"
```

3. 使用上一步中列出的环境变量启动应用程序。

在 Kubernetes 平台上启用 Application Signals

要启用在 Amazon EKS 以外的 Kubernetes 系统上运行的应用程序，请按照本节中的说明进行操作。Application Signals 同时支持 Java 应用程序和 Python 应用程序。

要求

- 您拥有启用 Application Signals 的 Kubernetes 集群的管理员权限。
- 您必须在运行 Kubernetes 集群的环境中安装 AWS CLI。有关安装 AWS CLI 的更多信息，请参阅安装或更新 AWS CLI 的最新版本。
- 您已在本地终端上安装了 kubectl 和 helm。有关更多信息，请参阅 kubectl 和 Helm 文档。

步骤 1：在账户中启用 Application Signals

如果您尚未在此账户中启用 Application Signals，则必须向 Application Signals 授予发现您的服务所需的权限。为此，请执行以下操作：您的账户只需执行一次该操作。

为您的应用程序启用 Application Signals

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务。
3. 选择开始发现您的服务。
4. 选中该复选框并选择开始发现您的服务。

   首次在您的账户中完成此步骤时，Application Signals 会创建 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色。此角色授予 Application Signals 以下权限：

   - `xray:GetServiceGraph`
   - `logs:StartQuery`
   - `logs:GetQueryResults`
   - `cloudwatch:GetMetricData`
   - `cloudwatch:ListMetrics`
   - `tag:GetResources`

   有关该角色的更多信息，请参阅 CloudWatch Application Signals 的服务相关角色权限。

步骤 2：在集群中安装 CloudWatch 代理 operator

安装 CloudWatch 代理 operator 会将 operator、CloudWatch 代理和其他自动检测工具安装到您的集群中。为此，请输入以下命令。将 *$REGION* 替换为您的 AWS 区域。将 *$YOUR_CLUSTER_NAME* 替换为要在 Application Signals 控制面板中为集群显示的名称。

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm install amazon-cloudwatch-operator aws-observability/amazon-cloudwatch-
observability \
--namespace amazon-cloudwatch --create-namespace \
--set region=$REGION \
--set clusterName=$YOUR_CLUSTER_NAME
```

有关更多信息，请参阅 GitHub 上的 amazon-cloudwatch-observability。

步骤 3：为 Kubernetes 集群设置 AWS 凭证

> **Important**
>
> 如果在 Amazon EC2 上托管您的 Kubernetes 集群，则可以跳过此部分并继续 步骤 4：添加注释。

如果在本地托管您的 Kubernetes 集群，则必须按照本节中的说明向您的 Kubernetes 环境添加 AWS 凭证。

设置本地 Kubernetes 集群的权限

1. 创建用于向本地主机提供权限的 IAM 用户：
   a. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
   b. 依次选择用户、创建用户。
   c. 在用户详情中，为用户名输入新 IAM 用户的名称。这是 AWS 的登录名，将用于对您的主机进行身份验证。然后选择下一个
   d. 在设置权限页面的权限选项下，选择直接附加策略。
   e. 从权限策略列表中，选择要添加到您的用户的 CloudWatchAgentServerPolicy 策略。然后选择下一步。
   f. 在查看并创建页面上，确保您对用户名满意，并且 CloudWatchAgentServerPolicy 策略位于权限摘要中。
   g. 选择 Create user。
2. 创建和检索您的 AWS 访问密钥和秘密密钥：
   a. 在 IAM 控制台的导航窗格中，选择用户，然后选择您在上一步中所创建用户的用户名。
   b. 在用户的页面上，选择安全凭证选项卡。然后，在访问密钥部分，选择创建访问密钥。
   c. 对于创建访问密钥步骤 1，选择命令行界面 (CLI)。
   d. 对于创建访问密钥步骤 2，（可选）输入标记，然后选择下一步。
   e. 对于创建访问密钥步骤 3，选择下载.csv 文件以保存包含您的 IAM 用户访问密钥和秘密访问密钥的 .csv 文件。您在后续步骤中需要此信息。
   f. 选择完成。

3. 通过输入以下命令，在本地主机中配置 AWS 凭证。将 *ACCESS_KEY_ID* 和 *SECRET_ACCESS_ID* 替换为您在上一步中下载的 .csv 文件中新生成的访问密钥和秘密访问密钥。默认情况下，凭证文件保存在 /home/*user*/.aws/credentials. 中

```
$ aws configure --profile AmazonCloudWatchAgent
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

4. 使用 Helm 图表编辑 CloudWatch 代理安装的自定义资源，以添加新创建的 AWS 凭证密钥。

```
kubectl edit amazoncloudwatchagent cloudwatch-agent -n amazon-cloudwatch
```

5. 当您的文件编辑器处于打开状态时，通过将以下配置添加到部署顶部，将 AWS 凭证挂载到 CloudWatch 代理容器中。将路径 /home/*user*/.aws/credentials 替换为本地 AWS 凭证文件的位置。

```
apiVersion: cloudwatch.aws.amazon.com/v1alpha1
kind: AmazonCloudWatchAgent
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  volumeMounts:
  - mountPath: /rootfs
    volumeMounts:
    - name: aws-credentials
      mountPath: /root/.aws
      readOnly: true
    volumes:
    - hostPath:
      path: /home/user/.aws/credentials
    name: aws-credentials
---
```

步骤 4：添加注释

下一步是通过向您的 Kubernetes 工作负载或命名空间添加特定于语言的注释，为 CloudWatch Application Signals 检测您的应用程序。此注释会自动检测应用程序以向 Application Signals 发送指标、跟踪和日志。

添加 Application Signals 的注释

1. 您有两种注释选项：

   - 为工作负载添加注释会自动检测集群中的单个工作负载。
   - 注释命名空间会自动检测所选命名空间中部署的所有工作负载。

   选择其中一个选项，然后按照相应的步骤操作。

2. 要注释单个工作负载，请输入以下命令之一。将 *$WORKLOAD_TYPE* 和 *$WORKLOAD_NAME* 替换为工作负载的实际值。

   - 对于 Java 工作负载：

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-java":
      "true"}}}}}'
     ```

   - 对于 Python 工作负载：

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-python":
      "true"}}}}}'
     ```

     对于 Python 应用程序，还需要额外的配置。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。

   - 对于 .NET 工作负载：

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-dotnet":
      "true"}}}}}'
     ```

     对于命名空间中的 .NET 工作负载：

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     dotnet=true
     ```

3. 要注释命名空间中的所有工作负载，请输入以下命令之一。将 *$NAMESPACE* 替换为您的命名空间的名称。

如果命名空间同时包含 Java 和 Python 工作负载，则将两种注释都添加到命名空间。

- 对于命名空间中的 Java 工作负载：

```
kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-java=true
```

- 对于命名空间中的 Python 工作负载：

```
kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
python=true
```

对于 Python 应用程序，还需要额外的配置。有关更多信息，请参阅 启用 Application Signals 后，Python 应用程序无法启动。

添加注释后，通过输入以下命令重新启动命名空间中的所有容器组（pod）：

```
kubectl rollout restart
```

4. 完成上述步骤后，在 CloudWatch 控制台中，依次选择 Application Signals、服务。这将打开控制面板，您可以在其中查看 Application Signals 收集的数据。可能需要几分钟才会显示数据。

   有关服务视图的更多信息，请参阅 使用 Application Signals 监控应用程序的运行状况。

更新至所需版本的代理或 Amazon EKS 加载项

2024 年 8 月 9 日之后，CloudWatch Application Signals 将不再支持旧版的 Amazon CloudWatch 可观测性 EKS 加载项、CloudWatch 代理和 AWS Distro for OpenTelemetry 自动检测代理。

- 对于 Amazon CloudWatch 可观测性 EKS 加载项，将不支持早于 `v1.7.0-eksbuild.1` 的版本。
- 对于 CloudWatch 代理，将不支持早于 `1.300040.0` 的版本。
- 对于 AWS Distro for OpenTelemetry 自动检测代理：
  - 对于 Java，将不支持早于 `1.32.2` 的版本。
  - 对于 Python，将不支持早于 `0.2.0` 的版本。

> **Important**
>
> 最新版本的代理包括对 Application Signals 指标架构的更新。这些更新不向后兼容，如果使用不兼容的版本，则可能会导致数据问题。为确保无缝过渡到新功能，请执行以下操作：
>
> - 如果您的应用程序在 Amazon EKS 上运行，则请务必在更新 Amazon CloudWatch 可观测性加载项后重新启动所有已检测的应用程序。
> - 对于在其他平台上运行的应用程序，请确保将 CloudWatch 代理和 AWS OpenTelemetry 自动检测代理同时升级到最新版本。

以下各节中的说明可以帮助您更新到支持的版本。

目录

- 更新 Amazon CloudWatch 可观测性 EKS 加载项
  - 使用控制台
  - 使用 AWS CLI
- 更新 CloudWatch 代理和 ADOT 代理
  - 在 Amazon ECS 上更新
  - 在 Amazon EC2 或其他架构上进行更新

更新 Amazon CloudWatch 可观测性 EKS 加载项

要更新 Amazon CloudWatch 可观测性 EKS 加载项，您可以使用 AWS Management Console 或 AWS CLI。

使用控制台

使用控制台升级加载项

1. 从以下位置打开 Amazon EKS 控制台：https://console.aws.amazon.com/eks/home#/clusters。
2. 选择要更新的 Amazon EKS 集群的名称。
3. 选择加载项选项卡，然后选择 Amazon CloudWatch 可观测性。
4. 依次选择编辑、要更新到的版本和保存更改。

   请务必选择 `v1.7.0-eksbuild.1` 或更高版本。

5. 输入以下 AWS CLI 命令之一以重新启动服务。

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

使用 AWS CLI

使用 AWS CLI 升级加载项

1. 输入以下命令以查找最新版本。

```
aws eks describe-addon-versions \
--addon-name amazon-cloudwatch-observability
```

2. 输入以下命令以更新加载项。将 *$VERSION* 替换为版本 v1.7.0-eksbuild.1 或更高版本。将 *$AWS_REGION* 和 *$CLUSTER* 替换为您的区域和集群名称。

```
aws eks update-addon \
--region $AWS_REGION \
--cluster-name $CLUSTER \
--addon-name amazon-cloudwatch-observability \
--addon-version $VERSION \
# required only if the advanced configuration is used.
--configuration-values $JSON_CONFIG
```

> **Note**
>
> 如果您为加载项使用自定义配置，则可以在 启用 CloudWatch Application Signals 中找到用于 *$JSON_CONFIG* 的配置示例。

3. 输入以下 AWS CLI 命令之一以重新启动服务。

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
```

```
# Restart a statefulset
kubectl rollout restart statefulset/name
```

更新 CloudWatch 代理和 ADOT 代理

如果您的服务在 Amazon EKS 以外的架构上运行，则需要同时升级 CloudWatch 代理和 ADOT 自动检测代理才能使用最新的 Application Signals 功能。

在 Amazon ECS 上更新

为在 Amazon ECS 上运行的服务升级您的代理

1. 创建新的任务定义修订。有关更多信息，请参阅使用控制台更新任务定义。
2. 将 `ecs-cwagent` 容器的 `$IMAGE` 替换为 Amazon ECR 上 cloudwatch-agent 的最新映像标签。

   如果您升级到固定版本，则请务必使用版本 `1.300040.0` 或更高版本。
3. 将 `init` 容器的 `$IMAGE` 替换为以下位置的最新映像标签：

   - 对于 Java，请使用 aws-observability/adot-autoinstrumentation-java

     如果您升级到固定版本，则请务必使用版本 `1.32.2` 或更高版本。
   - 对于 Python，请使用 aws-observability/adot-autoinstrumentation-python

     如果您升级到固定版本，则请务必使用版本 `0.2.0` 或更高版本。
4. 按照 步骤 4：使用 CloudWatch 代理检测您的应用程序 中的说明更新应用程序容器中的 Application Signals 环境变量。
5. 使用新任务定义部署服务。

在 Amazon EC2 或其他架构上进行更新

为在 Amazon ECS 或其他架构上运行的服务升级您的代理

1. 按照 下载 CloudWatch 代理软件包 中的说明操作，将 CloudWatch 代理升级到最新版本。请务必选择版本 `1.300040.0` 或更高版本。
2. 从以下位置之一下载最新版本的 AWS Distro for OpenTelemetry 自动检测代理：

   - 对于 Java，请使用 aws-otel-java-instrumentation

     如果您升级到固定版本，则请务必选择 `1.32.2` 或更高版本。

- 对于 Python，请使用 aws-otel-python-instrumentation

  如果您升级到固定版本，则请务必选择 `0.2.0` 或更高版本。

3. 将更新的 Application Signals 环境变量应用于您的应用程序，然后启动该应用程序。有关更多信息，请参阅 步骤 3：检测您的应用程序并将其启动。

对 Application Signals 安装进行问题排查

本节包含有关 CloudWatch Application Signals 的问题排查提示。

主题

- 启用 Application Signals 后，应用程序无法启动
- 启用 Application Signals 后，Python 应用程序无法启动
- Application Signals 控制面板中没有应用程序数据
- 服务指标或依赖项指标具有未知值
- 管理 Amazon CloudWatch Observability EKS 附加组件时处理 ConfigurationConflict
- 我想过滤掉不必要的指标和跟踪
- InternalOperation 表示什么？
- 如何为 .NET 应用程序启用日志记录？（预览版）
- 如何解决 .NET 应用程序中的程序集版本冲突？
- 我能禁用 FluentBit 吗？
- 是否可以在导出到 CloudWatch Logs 之前筛选容器日志？

启用 Application Signals 后，应用程序无法启动

如果您在 Amazon EKS 集群上启用 Application Signals 后，该集群上的应用程序仍未启动，请检查以下内容：

- 检查应用程序是否已被其他监控解决方案检测过。Application Signals 可能不支持与其他检测解决方案共存。
- 确认您的应用程序符合使用 Application Signals 的兼容性要求。有关更多信息，请参阅 Application Signals 支持的系统。
- 如果您的应用程序未能拉取 Application Signals 构件，例如 AWS Distro for OpenTelemetery Java 或 Python 代理和 CloudWatch 代理映像，则可能是网络问题。

要缓解此问题，请从应用程序部署清单中移除注释 `instrumentation.opentelemetry.io/inject-java: "true"` 或 `instrumentation.opentelemetry.io/inject-python: "true"`，然后重新部署您的应用程序。然后检查应用程序是否正在运行。

启用 Application Signals 后，Python 应用程序无法启动

OpenTelemetry 自动检测中的一个已知问题是，缺少 PYTHONPATH 环境变量有时会导致应用程序无法启动。要解决此问题，请确保将 PYTHONPATH 环境变量设置为应用程序工作目录的位置。有关此问题的更多信息，请参阅 Python autoinstrumentation setting of PYTHONPATH is not compliant with Python's module resolution behavior, breaking Django applications。

对于 Django 应用程序，还有其他必需的配置，这些配置在 OpenTelemetry Python 文档中进行了概述。

- 使用 `--noreload` 标志可防止自动重新加载。
- 将 `DJANGO_SETTINGS_MODULE` 环境变量设置为 Django 应用程序 `settings.py` 文件的位置。这样可确保 OpenTelemetry 能够正确访问您的 Django 设置，并与之集成。

Application Signals 控制面板中没有应用程序数据

如果 Application Signals 控制面板中缺少指标或跟踪，则可能是以下原因。只有在自上次更新以来等待 Application Signals 收集和显示数据的时间达到 15 分钟时，才调查这些原因。

- 确保您正在使用的库和框架受 ADOT Java 代理的支持。有关更多信息，请参阅库/框架。
- 确保 CloudWatch 代理正在运行。首先检查 CloudWatch 代理容器组（pod）的状态，并确保它们都处于 Running 状态。

  ```
  kubectl -n amazon-cloudwatch get pods.
  ```

  将以下内容添加到 CloudWatch 代理配置文件中以启用调试日志，然后重新启动代理。

  ```
  "agent": {

    "region": "${REGION}",
    "debug": true
  },
  ```

  然后检查 CloudWatch 代理容器组（pod）中是否存在错误。

- 检查 CloudWatch 代理的配置问题。确认以下内容仍在 CloudWatch 代理配置文件中，并且自添加以来，代理曾重新启动。

```
"agent": {
  "region": "${REGION}",
  "debug": true
},
```

  然后查看 OpenTelemetry 调试日志中是否有错误消息，例如 `ERROR io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export ...`。这些消息可能表明问题所在。

  如果这不能解决问题，请使用 `OTEL_` 命令描述容器组（pod），来转储并检查名称以 `kubectl describe pod` 开头的环境变量。

- 要启用 OpenTelemetry Python 调试日志记录，请将环境变量 `OTEL_PYTHON_LOG_LEVEL` 设置为 `debug` 并重新部署应用程序。
- 检查从 CloudWatch 代理导出数据的权限是否错误或不足。如果您在 CloudWatch 代理日志中看到 `Access Denied` 消息，则可能是问题所在。您安装 CloudWatch 代理时应用的权限后来可能被更改或撤销。
- 生成遥测数据时，请检查 AWS Distro for OpenTelemetry（ADOT）问题。

  确保检测注释 `instrumentation.opentelemetry.io/inject-java` 和 `sidecar.opentelemetry.io/inject-java` 已应用于应用程序部署，其值为 `true`。如果没有这些注释，即使 ADOT 附加组件安装正确，也不会对应用程序容器组（pod）进行检测。

  接下来，检查 `init` 容器是否已应用于应用程序且 `Ready` 状态为 `True`。如果 `init` 容器未准备就绪，请查看状态以了解原因。

  如果问题仍然存在，则请通过将环境变量 `OTEL_JAVAAGENT_DEBUG` 设置为 true 并重新部署应用程序来启用 OpenTelemetry Java SDK 的调试日志记录。然后查找以 `ERROR io.telemetry` 开头的消息。

- 指标/跨度导出程序可能正在删除数据。要找出答案，请查看应用程序日志中是否有包含 `Failed to export...` 的消息
- 向 Application Signals 发送指标或跨度时，CloudWatch 代理可能会受到限制。检查 CloudWatch 代理日志中是否有指示节流的消息。
- 确保您已启用服务发现设置。您只需在您的区域中执行一次此操作。

要进行确认，在 CloudWatch 控制台中，依次选择 Application Signals、服务。如果步骤 1 未标记为完成，则请选择开始发现您的服务。数据应在五分钟内开始流入。

服务指标或依赖项指标具有未知值

如果您在 Application Signals 控制面板中看到依赖项名称或操作的 UnknownService、UnknownOperation、UnknownRemoteService 或 UnknownRemoteOperation，则请检查未知远程服务和未知远程操作数据点的出现是否与其部署一致。

- UnknownService 表示检测的应用程序名称未知。如果 `OTEL_SERVICE_NAME` 环境变量未定义且未在 `OTEL_RESOURCE_ATTRIBUTES` 中指定 `service.name`，则服务名称将设置为 `UnknownService`。要解决此问题，请在 `OTEL_SERVICE_NAME` 或 `OTEL_RESOURCE_ATTRIBUTES` 中指定服务名称。
- UnknownOperation 表示所调用的操作名称未知。当 Application Signals 无法发现调用远程调用的操作名称，或者提取的操作名称包含高基数值时，就会发生这种情况。
- UnknownRemoteService 表示目标服务的名称未知。系统无法提取远程调用访问的目标服务名称时，就会发生这种情况。

  一种解决方案是围绕发送请求的函数创建一个自定义跨度，然后添加具有指定值的属性 `aws.remote.service`。另一种选择是配置 CloudWatch 代理以自定义 `RemoteService` 的指标值。有关 CloudWatch 代理中自定义的更多信息，请参阅 启用 CloudWatch Application Signals。
- UnknownRemoteOperation 表示目标操作的名称未知。系统无法提取远程调用访问的目标操作名称时，就会发生这种情况。

  一种解决方案是围绕发送请求的函数创建一个自定义跨度，然后添加具有指定值的属性 `aws.remote.operation`。另一种选择是配置 CloudWatch 代理以自定义 `RemoteOperation` 的指标值。有关 CloudWatch 代理中自定义的更多信息，请参阅 启用 CloudWatch Application Signals。

管理 Amazon CloudWatch Observability EKS 附加组件时处理 ConfigurationConflict

在安装或更新 Amazon CloudWatch Observability EKS 附加组件时，如果您发现故障是由 `ConfigurationConflict` 类型的 `Health Issue`（其描述以 `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode` 开头）引起的，这可能是因为您已经在集群上安装了 CloudWatch 代理及其相关组件，例如 ServiceAccount、ClusterRole 和

ClusterRoleBinding。当附加组件尝试安装 CloudWatch 代理及其关联组件时，如果检测到内容有任何变化，在默认情况下这会使安装或更新失败，以避免覆盖集群上资源的状态。

如果您正在尝试载入 Amazon CloudWatch Observability EKS 附加组件，但出现此故障，我们建议您删除之前在集群上安装的现有 CloudWatch 代理设置，然后安装 EKS 附加组件。请务必备份您可能对原始 CloudWatch 代理设置所做的任何自定义，例如自定义代理配置，并在下次安装或更新时将其提供给 Amazon CloudWatch Observability EKS 附加组件。如果您之前安装了 CloudWatch 代理以载入 Container Insights，请参阅 删除 Container Insights 的 CloudWatch 代理和 Fluent Bit 获取更多信息。

或者，该附加组件支持具有指定 `OVERWRITE` 功能的冲突解决配置选项。您可以使用此选项，通过覆盖集群上的冲突来继续安装或更新附加组件。如果您使用的是 Amazon EKS 控制台，则在创建或更新附加组件时选择可选配置设置时，会找到冲突解决方法。如果您使用的是 AWS CLI，则可以在命令中提供 `--resolve-conflicts OVERWRITE`，以创建或更新附加组件。

我想过滤掉不必要的指标和跟踪

如果 Application Signals 正在收集您不想要的跟踪和指标，则请参阅 管理高基数操作，了解有关使用自定义规则配置 CloudWatch 代理以减少基数的信息。

有关自定义跟踪采样规则的信息，请参阅 X-Ray 文档中的 Configure sampling rules。

`InternalOperation` 表示什么？

`InternalOperation` 表示由应用程序内部而不是外部调用触发的操作。看到 `InternalOperation` 是预期的正常行为。

以下是一些可以看到 `InternalOperation` 的典型示例：

- 启动时预加载：您的应用程序执行名为 `loadDatafromDB` 的操作，该操作在预热阶段从数据库中读取元数据。您不会将 `loadDatafromDB` 视为服务操作，而是将其归类为 `InternalOperation`。
- 后台异步执行：您的应用程序订阅事件队列，并在有更新时相应地处理流数据。每个触发的操作都将作为服务操作置于 `InternalOperation` 下。
- 从服务注册表检索主机信息：您的应用程序与服务注册表对话以进行服务发现。与发现系统的所有交互都归类为 `InternalOperation`。

如何为 .NET 应用程序启用日志记录？（预览版）

要为 .NET 应用程序启用日志记录，请配置以下环境变量。有关如何配置这些环境变量的更多信息，请参阅 OpenTelemetry 文档中的 .NET 自动仪表化问题的故障排除。

- OTEL_LOG_LEVEL
- OTEL_DOTNET_AUTO_LOG_DIRECTORY
- COREHOST_TRACE
- COREHOST_TRACEFILE

如何解决 .NET 应用程序中的程序集版本冲突？

如果您遇到以下错误，请参阅 OpenTelemetry 文档中的程序集版本冲突以了解解决步骤。

```
Unhandled exception. System.IO.FileNotFoundException: Could not load file or
 assembly 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'. The system cannot find the file
 specified.

File name: 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'
   at Microsoft.AspNetCore.Builder.WebApplicationBuilder..ctor(WebApplicationOptions
 options, Action`1 configureDefaults)
   at Microsoft.AspNetCore.Builder.WebApplication.CreateBuilder(String[] args)
   at Program.<Main>$(String[] args) in /Blog.Core/Blog.Core.Api/Program.cs:line 26
```

我能禁用 FluentBit 吗？

您可以通过配置 Amazon CloudWatch 可观测性 EKS 加载项来禁用 FluentBit。有关更多信息，请参阅（可选）其他配置。

是否可以在导出到 CloudWatch Logs 之前筛选容器日志？

不能，尚不支持筛选容器日志。

配置 Application Signals

本节包含有关配置 CloudWatch Application Signals 的信息。

跟踪采样率

默认情况下，启用 Application Signals 时，使用 `reservoir=1/s` 和 `fixed_rate=5%` 的默认采样率设置启用 X-Ray 集中采样。AWS Distro for OpenTelemetry（ADOT）SDK 代理的环境变量设置如下。

环境变量	值	备注
OTEL_TRACES_SAMPLER	xray	
OTEL_TRACES_SAMPLER_ARG	endpoint=http://cloudwatch-agent.amazon-cloudwatch:2000	CloudWatch 代理的端点

有关更改采样配置的信息，请参阅以下内容：

- 要更改 X-Ray 采样，请参阅 Configure sampling rules
- 要更改 ADOT 采样，请参阅 Configuring the OpenTelemetry Collector for X-Ray remote sampling

如果要禁用 X-Ray 集中采样并改用本地采样，请按如下方式为 ADOT SDK Java 代理设置以下值。以下示例将采样率设置为 5%。

环境变量	值
OTEL_TRACES_SAMPLER	parentbased_traceidratio
OTEL_TRACES_SAMPLER_ARG	0.05

有关更多高级采样设置的信息，请参阅 OTEL_TRACES_SAMPLER。

启用跟踪与日志关联

您可以在 Application Signals 中启用跟踪与日志关联。这会自动将跟踪 ID 和跨度 ID 注入到相关的应用程序日志中。然后，当您在 Application Signals 控制台中打开跟踪详细信息页面时，与当前跟踪相关的相关日志条目（如果有）会自动出现在页面底部。

例如，假设您注意到延迟图中出现了一个峰值。您可以选择图表上的点来加载该时间点的诊断信息。然后，您可以选择相关的跟踪以获取更多信息。查看跟踪信息时，可以向下滚动以查看与跟踪相关的日志。这些日志可能会显示与导致延迟峰值的问题相关的模式或错误代码。

为了实现跟踪日志关联，Application Signals 依赖于适用于 Java 的 Logger MDC 自动检测 和适用于 Python 的 OpenTelemetry Logging Instrumentation。两者均由 OpenTelemetry 社区提供。Application

Signals 使用此功能将跟踪 ID 和跨度 ID 等跟踪上下文注入应用程序日志。要启用此功能，您必须手动更改日志记录配置以启用自动检测。

根据应用程序运行的架构，除了执行本节中的步骤外，您可能还必须设置一个环境变量以启用跟踪日志关联。

- 在 Amazon EKS 上，无需采取进一步操作。
- 在 Amazon ECS 上，请参阅 步骤 4：使用 CloudWatch 代理检测您的应用程序 中过程的步骤 4 和步骤 5。
- 在 Amazon EC2 上，请参阅 步骤 3：检测您的应用程序并将其启动 中过程的步骤 4。

启用跟踪日志关联后，

跟踪日志关联设置示例

本节包含在多个环境中设置跟踪日志关联的示例。

Spring Boot for Java

假设您在名为 `custom-app` 的文件夹中有一个 Spring Boot 应用程序。应用程序配置通常是一个名为 `custom-app/src/main/resources/application.yml` 的 YAML 文件，可能如下所示：

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...
```

要启用跟踪日志关联，请添加以下日志记录配置。

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...

logging:
```

```
  pattern:
    level: trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags}
 %5p
```

Logback for Java

在日志记录配置（例如 logback.xml）中，将跟踪上下文 `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` 插入到编码器的 `pattern` 中。例如，以下配置在日志消息之前添加跟踪上下文。

```
<appender name="FILE" class="ch.qos.logback.core.FileAppender">
  <file>app.log</file>
  <append>true</append>
  <encoder>
    <pattern>trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n</pattern>
  </encoder>
</appender>
```

有关 Logback 中编码器的更多信息，请参阅 Logback 文档中的 Encoders。

Log4j2 for Java

在日志记录配置（例如 log4j2.xml）中，将跟踪上下文 `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` 插入到 `PatternLayout` 中。例如，以下配置在日志消息之前添加跟踪上下文。

```
<Appenders>
  <File name="FILE" fileName="app.log">
    <PatternLayout pattern="trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n"/>
  </File>
</Appenders>
```

有关 Log4j2 中模式布局的更多信息，请参阅 Log4j2 文档中的 Pattern Layout。

Log4j for Java

在日志记录配置（例如 log4j.xml）中，将跟踪上下文 `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` 插入到 `PatternLayout` 中。例如，以下配置在日志消息之前添加跟踪上下文。

```
<appender name="FILE" class="org.apache.log4j.FileAppender">;
  <param name="File" value="app.log"/>;
  <param name="Append" value="true"/>;
  <layout class="org.apache.log4j.PatternLayout">;
    <param name="ConversionPattern" value="trace_id=%mdc{trace_id} span_id=
%mdc{span_id} trace_flags=%mdc{trace_flags} %5p - %m%n"/>;
  </layout>;
</appender>;
```

有关 Log4j 中模式布局的更多信息，请参阅 Log4j 文档中的 Class Pattern Layout。

Python

运行应用程序时，将环境变量 OTEL_PYTHON_LOG_CORRELATION 设置为 true。有关更多信息，请参阅 Python OpenTelemetry 文档中的 Enable trace context injection。

启用指标与日志关联

如果您将应用程序日志发布到 CloudWatch Logs 中的日志组，则可以在 Application Signals 中启用指标与应用程序日志关联。通过指标日志关联，Application Signals 控制台会自动显示与指标关联的相关日志组。

例如，假设您注意到延迟图中出现了一个峰值。您可以选择图表上的点来加载该时间点的诊断信息。诊断信息将显示与当前服务和指标关联的相关应用程序日志组。然后，您可以选择一个按钮对这些日志组运行 CloudWatch Logs Insights 查询。根据应用程序日志中包含的信息，这可能有助于您调查延迟峰值的原因。

根据应用程序运行的架构，您可能还必须设置一个环境变量以启用指标与应用程序日志关联。

- 在 Amazon EKS 上，无需采取进一步操作。
- 在 Amazon ECS 上，请参阅 步骤 4：使用 CloudWatch 代理检测您的应用程序 中过程的步骤 4 和步骤 5。
- 在 Amazon EC2 上，请参阅 步骤 3：检测您的应用程序并将其启动 中过程的步骤 4。

管理高基数操作

Application Signals 包括 CloudWatch 代理中的设置，您可以使用这些设置来管理操作的基数和管理指标导出，以优化成本。默认情况下，当服务的不同操作数随着时间的推移超过默认阈值 500 时，指标限制功能将变为活动状态。您可以通过调整配置设置来调整行为。

确定指标限制是否已激活

您可以使用以下方法了解是否发生默认指标限制。如果是，应考虑执行下一节中的步骤，优化基数控制。

- 在 CloudWatch 控制台中，依次选择 Application Signals、服务。如果您看到名为 AllOtherOperations 的操作或名为 AllOtherRemoteOperations 的远程操作，则表示发生指标限制。
- 如果 Application Signals 收集的任何指标其 `Operation` 维度的值均为 `AllOtherOperations`，则说明发生指标限制。
- 如果 Application Signals 收集的任何指标其 `RemoteOperation` 维度的值均为 `AllOtherRemoteOperations`，则说明发生指标限制。

优化基数控制

要优化基数控制，可以执行以下操作：

- 创建用于聚合操作的自定义规则。
- 配置您的指标限制策略。

创建用于聚合操作的自定义规则

高基数操作有时可能是由从上下文中提取的不当唯一值引起的。例如，发出路径中包含用户 ID 或会话 ID 的 HTTP/S 请求可能会导致数百个不同的操作。要解决此类问题，建议您使用自定义规则配置 CloudWatch 代理，以重写这些操作。

如果通过单个 `RemoteOperation` 调用（例如 `PUT /api/customer/owners/123`、`PUT /api/customer/owners/456` 和类似的请求）生成大量不同的指标激增，则建议您将这些操作合并到一个 `RemoteOperation` 中。一种方法是将所有以 `PUT /api/customer/owners/` 开头的 `PUT /api/customer/owners/{ownerId}` 调用标准化为统一的格式，尤其是 `RemoteOperation`。以下示例对此进行了说明。有关其他自定义规则的信息，请参阅 启用 CloudWatch Application Signals。

```
{
   "logs":{
      "metrics_collected":{
         "application_signals":{
            "rules":[
               {
                  "selectors":[
                     {
```

```
                        "dimension":"RemoteOperation",
                        "match":"PUT /api/customer/owners/*"
                     }
                  ],
                  "replacements":[
                     {
                        "target_dimension":"RemoteOperation",
                        "value":"PUT /api/customer/owners/{ownerId}"
                     }
                  ],
                  "action":"replace"
               }
            ]
         }
      }
   }
}
```

在其他情况下，高基数指标可能已汇总到 AllOtherRemoteOperations，并且可能不清楚包含哪些具体指标。CloudWatch 代理能够记录已删除的操作。要确定已删除的操作，请使用以下示例中的配置激活日志记录，直到问题再次出现。然后检查 CloudWatch 代理日志（可通过容器 stdout 或 EC2 日志文件访问）并搜索关键字 drop metric data。

```
{
  "agent": {
    "config": {
      "agent": {
        "debug": true
      },
      "traces": {
        "traces_collected": {
          "application_signals": {
          }
        }
      },
      "logs": {
        "metrics_collected": {
          "application_signals": {
            "limiter": {
              "log_dropped_metrics": true
            }
          }
        }
```

```
      }
    }
  }
}
```

创建指标限制策略

如果默认指标限制配置无法解决服务的基数，则可以自定义指标限制器配置。为此，请在 CloudWatch 代理配置文件中的 `logs/metrics_collected/application_signals` 部分下添加 `limiter` 部分。

以下示例将指标限制阈值从 500 个不同指标降低到 100 个。

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "limiter": {
          "drop_threshold": 100
        }
      }
    }
  }
}
```

使用 Application Signals 监控应用程序的运行状况

使用 CloudWatch 控制台中的 Application Signals 监控应用程序的运行状况并对其进行问题排查：

- 监控您的应用程序服务：作为日常运行监控的一部分，使用服务页面查看完整服务摘要。查看故障率最高或延迟时间最长、服务级别指标（SLI）运行不正常的服务。选择一项服务即可打开服务详细信息页面，查看详细指标、服务运营、Synthetics Canary、和客户端请求。这一操作可以帮助您进行问题排查并确定运行问题的根本原因。
- 检查您的应用程序拓扑：使用服务地图了解和监控您的应用程序拓扑一段时间内的变化，包括客户端、Synthetics Canary、服务和依赖项之间的关系。即时查看服务级别指标（SLI）运行状况以及调用量、故障率和延迟等关键指标。在服务详细信息页面深入查看更多详细信息。

探索示例场景，该场景演示如何使用这些页面快速排查运营服务运行状况问题，覆盖初始检测到确定根本原因的全过程。

Application Signals 如何实现运行状况监控

为 Application Signals 启用应用程序后，您的应用程序服务、API 及其依赖项将被自动发现并显示在服务、服务详细信息和服务地图页面。Application Signals 从多个来源收集信息，进而实现服务发现和运行状况监控：

- AWS Distro for OpenTelemetry（ADOT）：作为启用 Application Signals 的其中一步，将 OpenTelemetry Java 和 Python 自动检测库配置为发出 CloudWatch 代理收集的指标和跟踪。这些指标和跟踪用于发现服务、操作、依赖项和其他服务信息。
- 服务级别目标（SLO）：为服务创建服务级别目标后，“服务”、“服务详细信息”和“服务地图”页面会显示服务级别指标（SLI）的运行状况。SLI 可以监控延迟、可用性和其他运行指标。
- CloudWatch Synthetics Canary：当您为 Canary 配置 X-Ray 跟踪时，Canary 脚本的服务调用将与您的服务关联并显示在服务详细信息页面。
- CloudWatch 真实用户监控（RUM）：在您的 CloudWatch RUM Web 客户端上启用 X-Ray 跟踪后，服务请求会自动关联并显示在“服务详细信息”页面。
- AWS Service Catalog AppRegistry：Application Signals 会自动发现您账户中的 AWS 资源，并允许您将其归为 AppRegistry 中创建的逻辑应用程序。“服务”页面显示的应用程序名称基于运行服务的相关计算资源。

> **Note**
>
> Application Signals 根据所选当前时间筛选器中发出的指标和跟踪显示您的服务和操作。（默认为过去三个小时。）如果用于服务、运营、依赖项、Synthetics Canary 或客户端页面的当前时间筛选器内并无活动，则不予显示。
> 目前最多可以显示 1000 项服务。服务和服务拓扑的发现最多可能会延迟 10 分钟。对服务级别指标（SLI）运行状况的评估最多可能延迟 15 分钟。

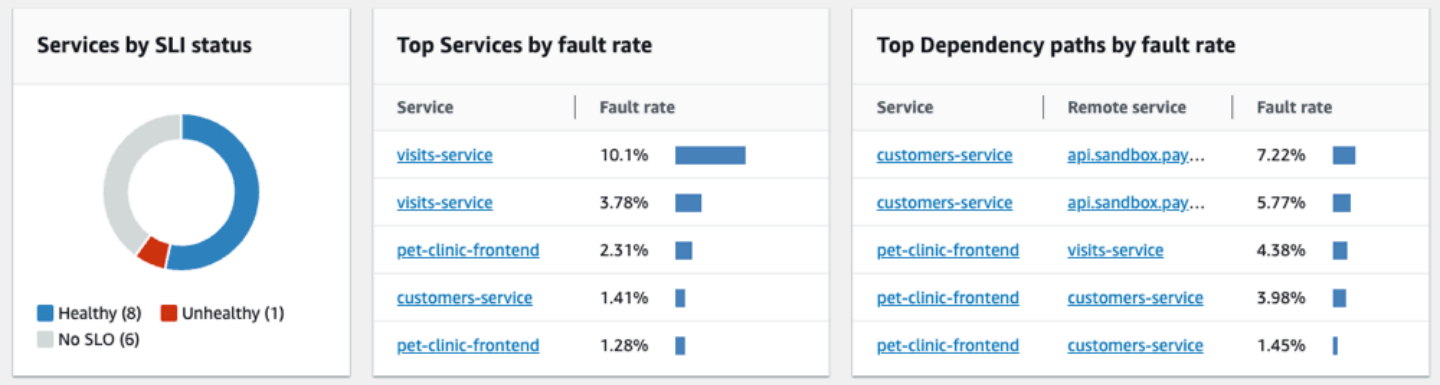
通过“服务”页面查看整体服务活动和运行状况

通过“服务”页面查看已为 Application Signals 启用的服务列表。您还可以查看运行指标，并快速浏览哪些服务具有运行不正常的服务级别指标（SLI）。应深入调查性能异常，并确定操作问题的根本原因。要查看此页面，请打开 CloudWatch 控制台，然后选择左侧导航窗格的 Application Signals部分下的服务。

探索服务的运行状况指标

“服务”页面顶部包括一个整体服务运行状况图和几个表格，这些表格显示按故障率排序的顶部服务和服务依赖项。左侧的“服务”图表显示在当前页面级时间筛选条件期间，运行正常或运行不正常的服务级别指标（SLI）的服务数量明细。SLI 可以监控延迟、可用性和其他运行指标。

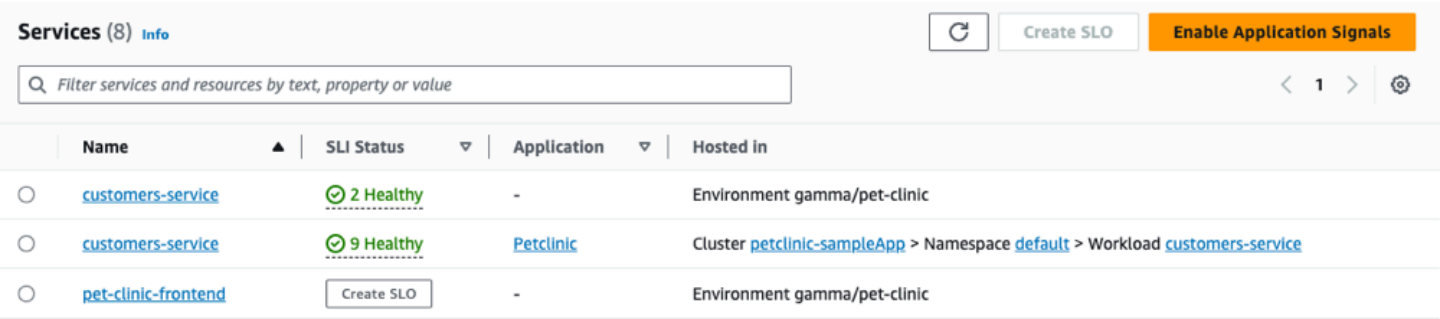
图表旁边的两个表格显示按故障率排序的顶部服务列表。选择任一表格中的任意服务名称，打开服务详细信息页面并查看服务操作详细信息。选择依赖项路径，打开详细信息页面并查看服务依赖项详细信息。即使在页面右上角选择更长时间段的筛选条件，这两个表格也最多只能显示过去三个小时内的信息。



通过“服务”表格监控运行状况

“服务”表格显示已为 Application Signals 启用的服务列表。选择启用 Application Signals，打开设置页面并开始配置服务。有关更多信息，请参阅启用 Application Signals。

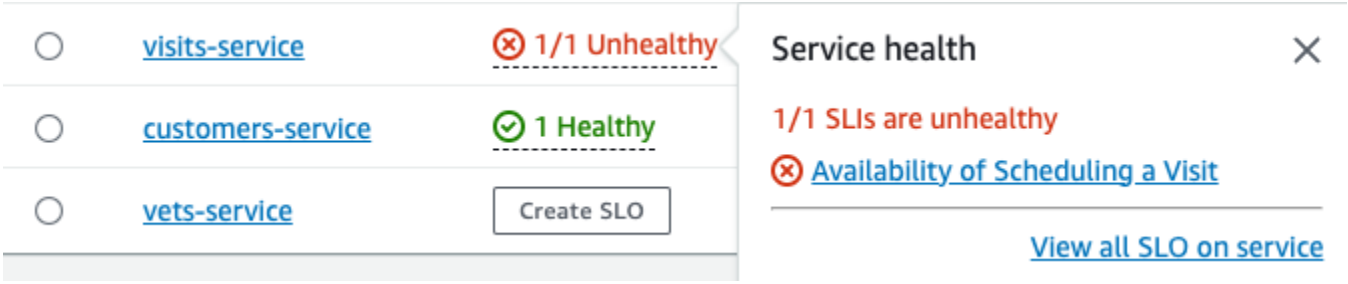
通过从筛选条件文本框中选择一个或多个属性来筛选“服务”表格，可便于查找所需内容。选择各个属性时，系统将引导您选择筛选条件。您将在筛选条件文本框下看到完整的筛选条件。随时选择清除筛选条件以移除表格筛选条件。



选择表格中任何服务名称，查看包含服务级别指标、操作和其他详细信息的服务详细信息页面。如果您已将服务的底层计算资源与 AppRegistry 中的应用程序或 AWS Management Console 主页上的“应用程序”卡相关联，则可选择应用程序名称，以在 myApplications 控制台页面中显示应用程序详细信

息。对于托管在 Amazon EKS 中的服务，选择托管在列中的任意链接，查看 CloudWatch Container Insights 中的集群、命名空间或工作负载。对于在 Amazon ECS 或 Amazon EC2 上运行的服务，将显示环境值。

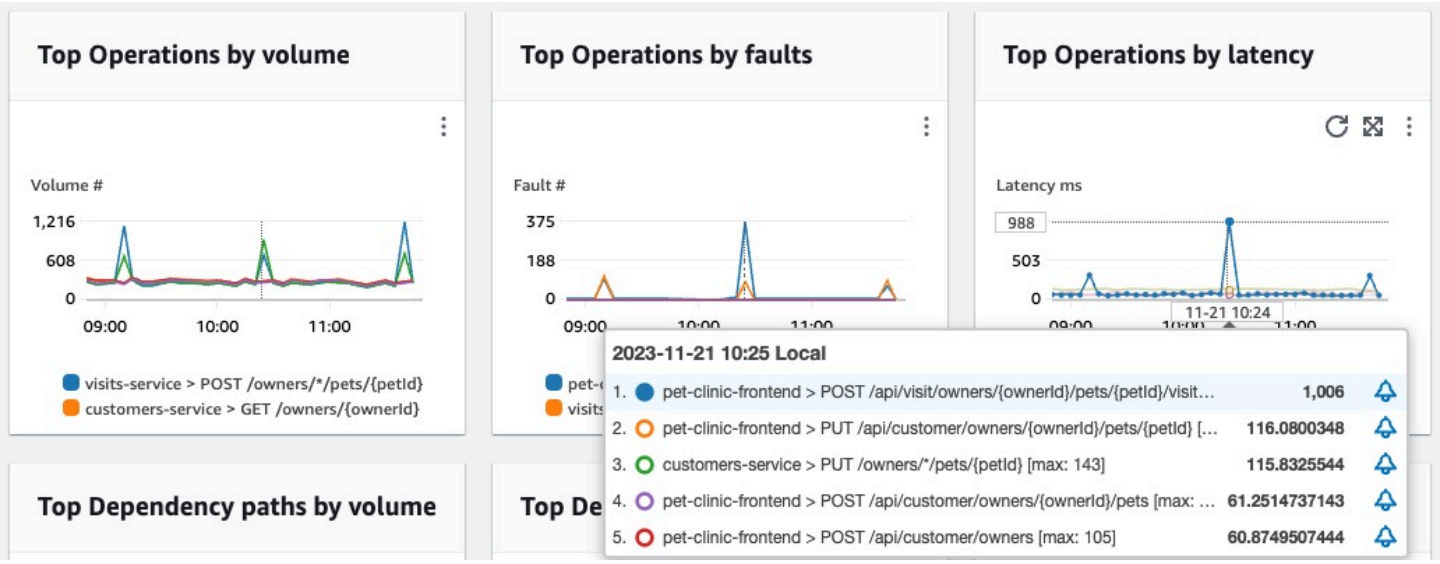
表格显示了每项服务的服务级别指标（SLI）状态。选择服务的 SLI 状态以显示弹出窗口，该弹出窗口包含指向任何运行不正常的 SLI 的链接，以及用于查看该服务的所有 SLO 的链接。



如果尚未为某项服务创建 SLO，请选择 SLI 状态列中的创建 SLO 按钮。要为任何服务额外创建 SLO，请选择服务名称旁边的选项按钮，然后选择表格右上角的创建 SLO。创建 SLO 时，您可以一目了然地看到运行正常和运行不正常的服务和操作。有关更多信息，请参阅服务级别目标（SLO）。

查看热门操作和依赖项指标

在“服务”表格下方，您可以按调用量、故障和延迟查看所有服务的顶部操作和依赖项。这组图表提供了有关所有服务中哪些操作或依赖项可能运行不正常的重要信息。选择图表中的任意点，即可查看包含详细序列信息的弹出窗口。将鼠标悬停在图表底部的序列描述上，即可看到一个弹出窗口，其中包含特定操作或依赖项路径的详细指标。选择图表右上角的快捷菜单按钮，可以查看其他选项，包括查看 CloudWatch 指标或日志页面。



通过服务详细信息页面查看详细的服务活动和运行状况

当您分析应用程序时，Amazon CloudWatch Application Signals 会映射您的应用程序发现的所有服务。使用服务详细信息页面查看单个服务的服务概述、操作、依赖项、Canary 和客户端请求。要查看服务详细信息页面，请执行以下操作：

- 打开 CloudWatch 控制台。
- 在左侧导航窗格的 Application Signals 部分下选择服务。
- 从服务、热门服务或依赖项表中选择任何服务的名称。

服务详细信息页面分为以下选项卡：

- 概述 – 使用此选项卡查看单个服务概述，包括操作数量、依赖项、Synthetics 和客户端页面。该选项卡显示整个服务、顶部操作和依赖项的关键指标。这些指标包括有关该服务所有服务操作的延迟、故障和错误的时间序列数据。
- 服务操作 — 使用此选项卡查看服务调用的操作列表，以及包含衡量每个操作运行状况关键指标的交互式图表。您可以在图表中选择一个数据点，以获取与该数据点关联的跟踪、日志或指标的相关信息。
- 依赖项 — 使用此选项卡查看服务调用的依赖项列表，以及依赖项指标列表。
- Synthetics Canary — 使用此选项卡可查看模拟用户对您的服务的调用的 Synthetics Canary 列表，以及这些 Canary 如何调用的关键性能指标。
- 客户端页面 — 使用此选项卡查看调用服务的客户端页面列表，以及衡量客户端与应用程序交互质量的指标。

查看服务概述

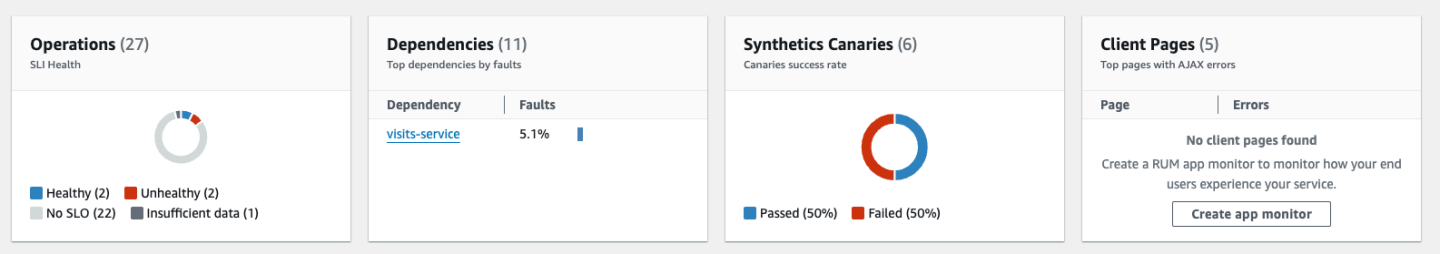
使用服务概述页面，可以在一个位置查看所有服务操作指标的高度概括。检查与您的应用程序交互的所有操作、依赖项、客户端页面和 Synthetics Canary 的性能。使用此信息可以帮助您确定在何处集中精力来识别问题、排除错误和找到进行优化的机会。

选择服务详细信息中的任何链接，以查看与特定服务相关的信息。例如，对于托管在 Amazon EKS 中的服务，服务详细信息页面会显示集群、命名空间和工作负载信息。对于托管在 Amazon ECS 或 Amazon EC2 中的服务，服务详细信息页面将显示环境值。

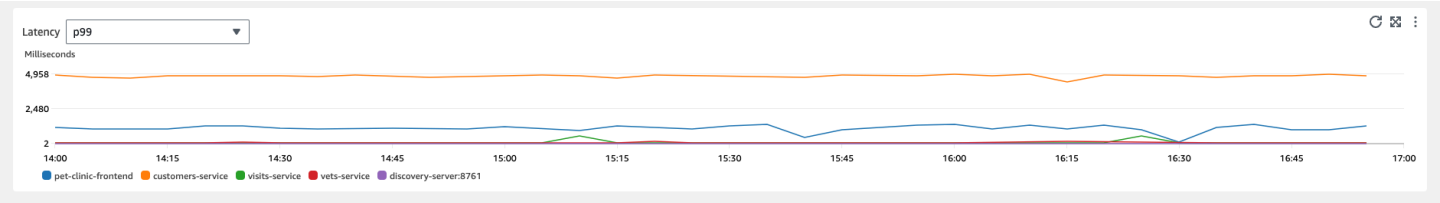
在服务下，概述选项卡显示以下各项的摘要：

- 操作 – 使用此选项卡查看服务操作的运行状况。运行状况由服务级别指标（SLI）确定，而这些指标被定义为服务级别目标（SLO）的一部分。
- 依赖项 – 使用此表查看您的应用程序调用的服务的主要依赖项，按故障率列出。
- Synthetics Canary – 使用此选项卡查看对与您的服务关联的端点或 API 的模拟调用结果，以及失败的 Canary 数量。
- 客户页面 – 使用此选项卡查看出现异步 JavaScript and XML（AJAX）错误的客户端调用的热门页面。

下图显示了您的服务概述：



概述选项卡还显示了所有服务中延迟时间最高的依赖项的图表。使用 p99、p90 和 p50 延迟指标来快速评估哪些依赖项导致了总服务延迟，如下所示：

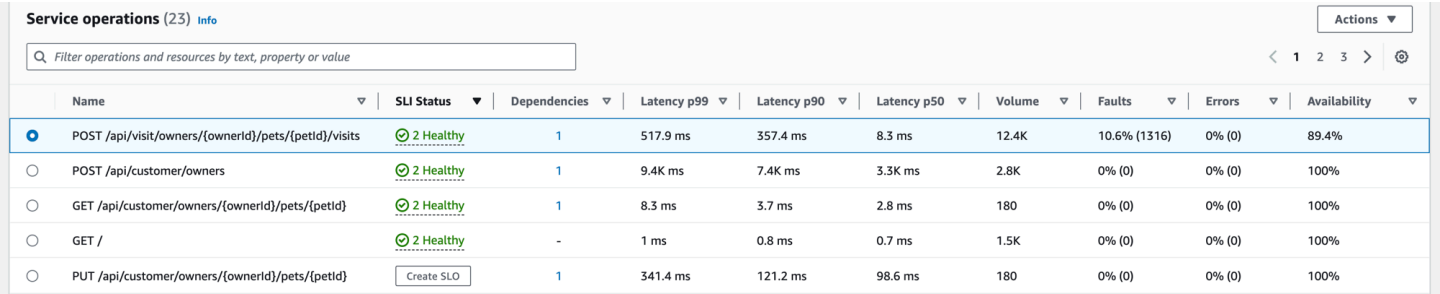


例如，上图显示，向客户服务依赖项发出的请求中有 99% 在约 4950 毫秒内完成。其他依赖项花费的时间更少。

按延迟显示前四项服务操作的图表显示了这些服务的请求量、可用性、故障率和错误率，如下图所示：

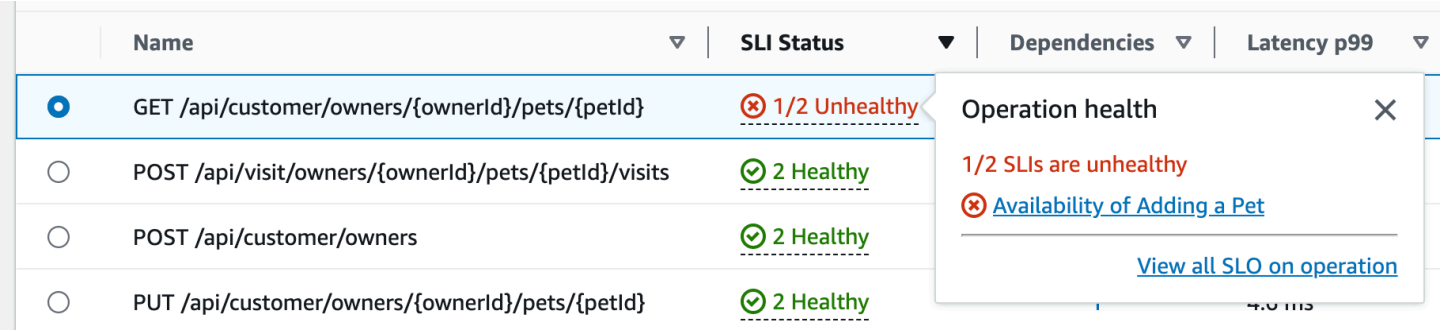
查看您的服务操作

当您分析应用程序时，Application Signals 会发现您的应用程序调用的所有服务操作。使用服务操作选项卡，查看包含服务操作和衡量所选操作性能的一组指标的表。这些指标包括 SLI 状态、依赖项数量、延迟、操作量、故障、错误和可用性，如下图所示：



通过从筛选条件文本框中选择一个或多个属性来筛选表格，更易于查找服务操作。选择各个属性时，您将在系统引导下选择筛选条件，并在筛选条件文本框下看到完整的筛选条件。随时选择清除筛选条件以移除表格筛选条件。

选择操作的 SLI 状态以显示弹出窗口，该弹出窗口包含指向任何运行不正常的 SLI 的链接，以及用于查看该操作的所有 SLO 的链接，如下表中所示：



服务操作表列出每项操作的 SLI 状态、运行正常或不正常的 SLI 数量以及 SLO 总数。

使用 SLI 监控延迟、可用性和其他操作指标，从而衡量服务的运行状况。使用 SLO 检查服务和操作的性能和运行状况。

要创建 SLO，可以执行以下操作：

- 如果某项操作没有 SLO，请选择 SLI 状态列中的创建 SLO 按钮。
- 如果某项操作已有 SLO，请执行以下操作：
  - 选中操作名称旁的单选按钮。
  - 从表格右上角的操作向下箭头中选择创建 SLO。

有关更多信息，请参阅 service level objectives (SLOs)。

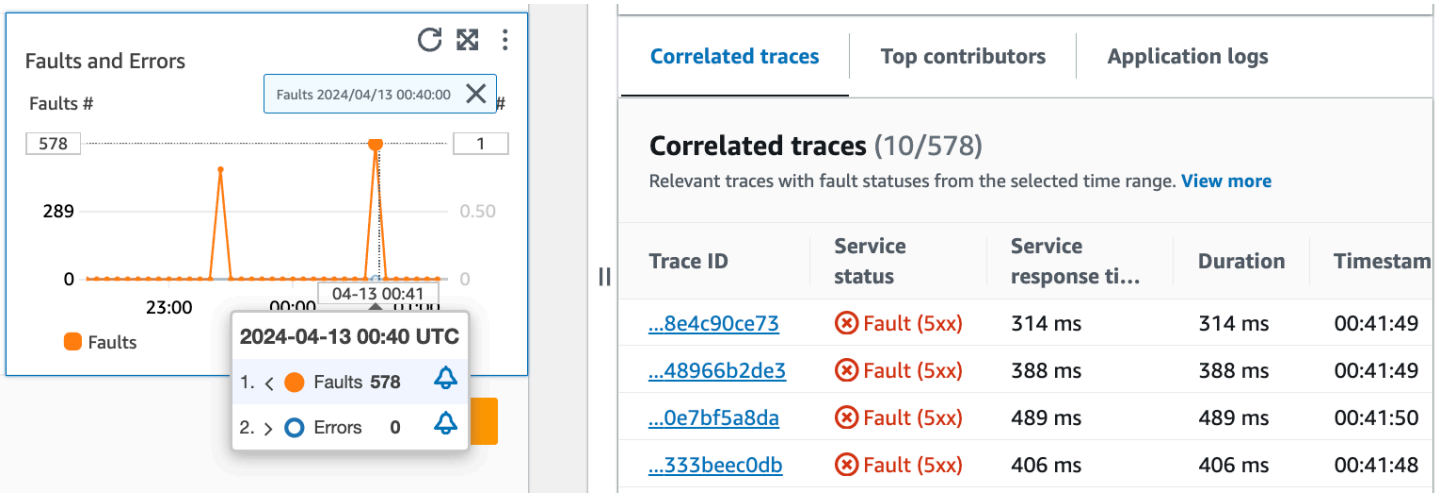
依赖项列显示了此操作调用的依赖项数量。选择此数字可打开筛选到选定操作的依赖项选项卡。

查看服务操作指标、相关跟踪和应用程序日志

Application Signals 将服务操作指标与 AWS X-Ray 跟踪、CloudWatch Container Insights 和应用程序日志关联。使用这些指标排查运行状况问题。要将指标作为图形信息查看，请执行以下操作：

1. 选择服务操作表中的一项服务操作，以查看表格上方所选操作的一组图表，其中包含操作量和可用性、延迟以及故障和错误的指标。
2. 将鼠标悬停在图表中的某个点上，即可查看更多信息。
3. 选择一个点即可打开诊断窗格，其中显示图表中选定点的相关跟踪、指标和应用程序日志。

下图显示了将鼠标悬停在图表中的某个点上后出现的工具提示，以及单击某个点后出现的诊断窗格。工具提示包含有关故障和错误图表中关联数据点的信息。该窗格包含与选定点关联的相关跟踪、排名靠前的贡献者和应用程序日志。



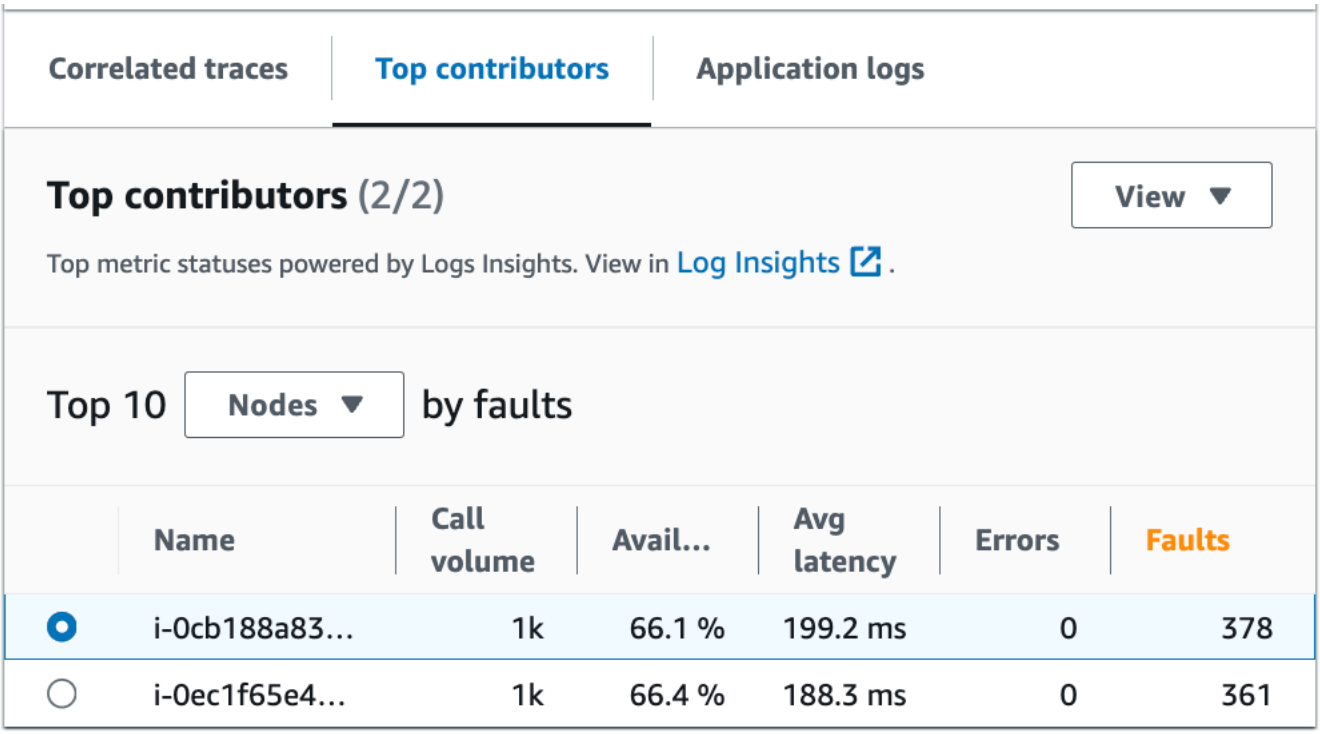
相关跟踪

查看相关跟踪以了解跟踪的潜在问题。您可以检查相关的跟踪或与之关联的任何服务节点的行为是否类似。要检查相关跟踪，从相关跟踪表中选择一个跟踪 ID，以打开所选跟踪的 X-Ray 跟踪详细信息页面。跟踪详细信息页面包含与所选跟踪关联的服务节点图和跟踪分段的时间表。

排名靠前的贡献者

查看主要贡献者，以查找指标的主要输入来源。按不同的组成部分对促成因素进行分组，以寻找组内的相似之处，并了解他们之间的跟踪行为有何不同。

主要贡献者选项卡提供了每个组的调用量、可用性、平均延迟、错误和故障等指标。下面的示例图显示了，部署在 Amazon EKS 平台上的应用程序的一组指标的主要贡献者：



主要贡献者包含以下指标：

- 调用量 - 使用调用量来了解一个组在每个时间间隔的请求数。
- 可用性 - 使用可用性查看组中未检测到故障的时间百分比。
- 平均延迟 - 使用延迟来检查某个组在某个时间间隔内运行请求的平均时间，该时间间隔取决于您正在调查的请求是在多久之前发出的。对于不足 15 天前提出的请求，将以 1 分钟为间隔进行评估。对于 15 天到 30 天前（含 15 天和 30 天）提出的请求，将以 5 分钟为间隔进行评估。例如，如果您调查 15 天前导致故障的请求，则呼叫量指标等于每 5 分钟间隔的请求数。
- 错误 - 在一段时间间隔内测量的每个组的错误数。
- 故障 - 一段时间间隔内每个组的故障数。

使用 Amazon EKS 或 Kubernetes 的主要贡献者

使用部署在 Amazon EKS 或 Kubernetes 上的应用程序的主要贡献者相关信息查看按节点、容器组（pod）和 PodTemplateHash 分组的运行状况指标。以下定义适用：

- 容器组（pod）是一组共享存储和资源的一个或多个 Docker 容器。容器组（pod）是可以部署在 Kubernetes 平台上的最小单元。按容器组（pod）分组以检查错误是否与容器组（pod）特定的限制有关。
- 节点是运行容器组（pod）的服务器。按节点分组以检查错误是否与节点特定的限制有关。

- 容器组（pod）模板哈希用于查找部署的特定版本。按容器组（pod）模板哈希分组以检查错误是否与特定部署有关。

使用 Amazon EC2 的主要贡献者

使用部署在 Amazon EKS 上的应用程序的主要贡献者相关信息，查看按实例 ID 和自动扩缩组来分组的运行状况指标。以下定义适用：

- 实例 ID 是您的服务运行的 Amazon EC2 实例的唯一标识符。按实例 ID 分组以检查错误是否与特定 Amazon EC2 实例有关。
- 自动扩缩组是 Amazon EC2 实例的集合，允许您纵向扩展或缩减服务应用程序请求所需的资源。如果您想检查错误的范围是否仅限于组内的实例，则请按自动扩缩组进行分组。

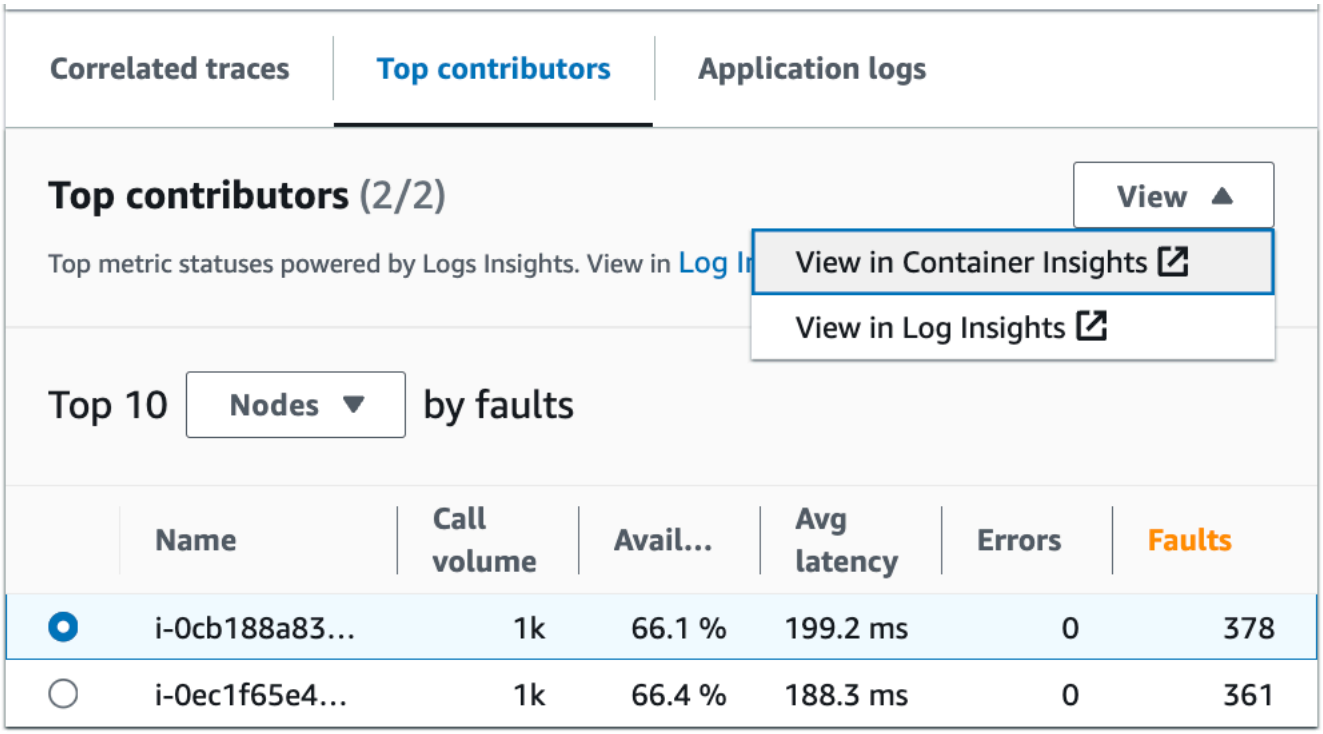
使用自定义平台的主要贡献者

对于使用自定义工具部署的应用程序，使用其主要贡献者的相关信息查看按主机名分组的运行状况指标。以下定义适用：

- 主机名标识连接到网络的设备，例如端点或 Amazon EC2 实例。按主机名分组以检查您的错误是否与特定物理设备或虚拟设备有关。

查看 Log Insights 和 Container Insights 中的主要贡献者

查看在 Log Insights 中为您的主要贡献者生成指标的自动查询，并对其进行修改。在 Container Insights 中按特定组 [例如容器组（pod）或节点] 查看基础设施性能指标。您可以按资源消耗对集群、节点或工作负载进行分类，并在最终用户体验受到影响之前快速识别异常或/和主动降低风险。显示如何选择这些选项的图像如下：



在 Container Insights 中，您可以查看特定于排名靠前的贡献者分组的 Amazon EKS 或 Amazon ECS 容器的指标。例如，如果您按容器组（pod）对 EKS 容器进行分组以生成排名靠前的贡献者，则 Container Insights 将显示针对您的容器组（pod）筛选的指标和统计数据。

在 Log Insights 中，您可以使用以下步骤修改在排名靠前的贡献者下生成指标的查询：

1. 选择在 Log Insights 中查看。打开的 Logs Insights 页面包含自动生成的查询，并包含以下信息：

   - 日志集群组名称。
   - 您使用 CloudWatch 调查的操作。
   - 图表上与之交互的操作运行状况指标的汇总。

   系统会自动筛选日志结果，以显示您在服务图表上选择数据点之前最后五分钟内的数据。
2. 要编辑查询，请将生成的文本替换为您更改的内容。您还可以使用查询生成器来帮助您生成新查询或更新现有查询。

应用程序日志

使用应用程序日志选项卡中的查询为您当前的日志组、服务生成记录的信息，并插入时间戳。日志组是您在配置应用程序时可以定义的一组日志流。

使用日志组来组织具有类似特征的日志，其中包括以下内容：

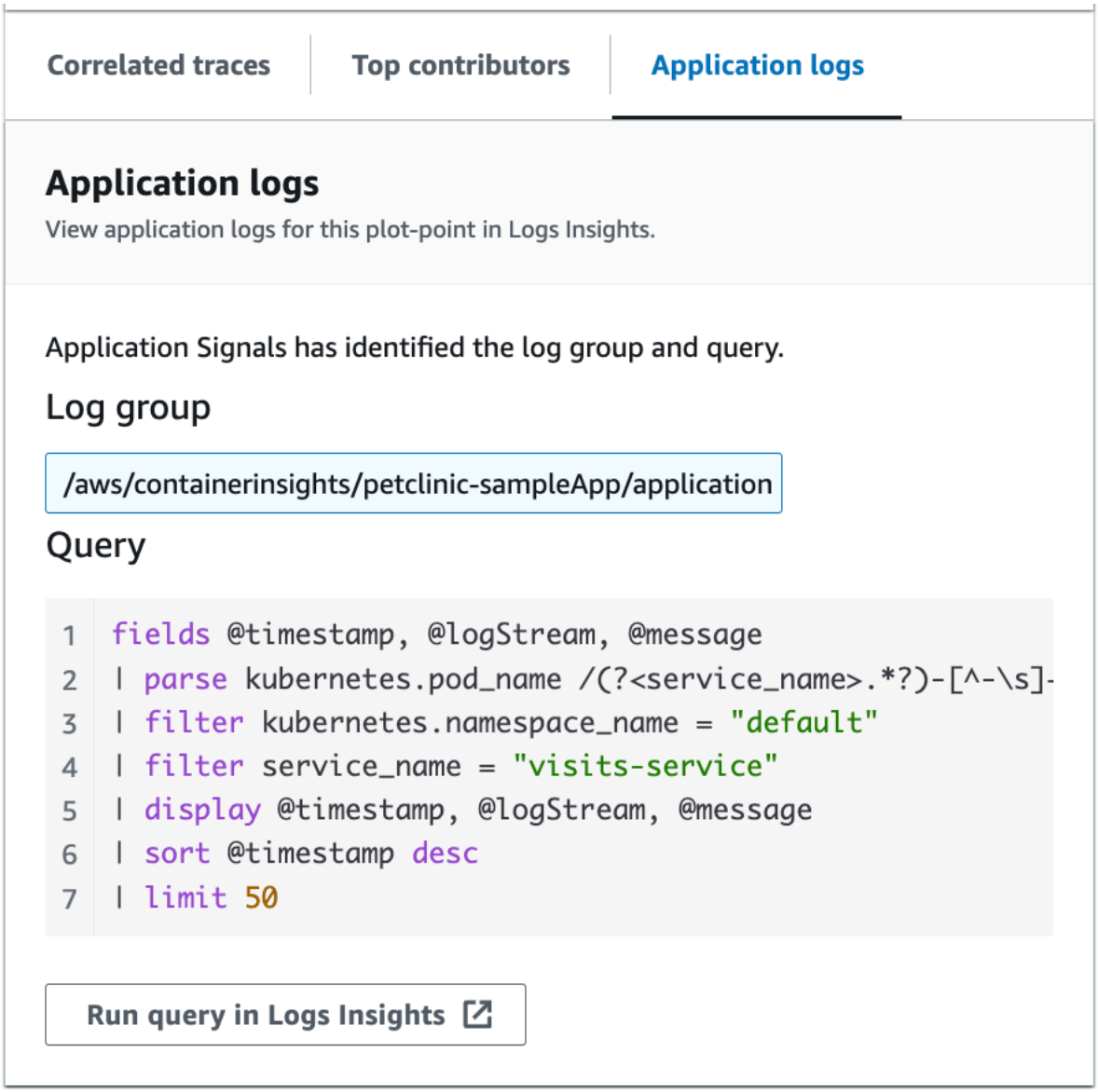
- 捕获来自特定组织、来源或功能的日志。
- 捕获由特定用户访问的日志。
- 捕获特定时间段的日志。

使用这些日志流来跟踪特定组或时间范围。您还可以为这些日志组设置监控规则、警报和通知。有关日志组的更多信息，请参阅使用日志组和日志流。

应用程序日志查询会返回日志组的日志、重复出现的文本模式和图形可视化内容。

要运行查询，请选择在 Logs Insights 中运行查询，以运行自动生成的查询或修改查询。要编辑查询，请将自动生成的文本替换为您更改的内容。您还可以使用查询生成器来帮助您生成新查询或更新现有查询。

下图显示根据服务操作图中所选点自动生成的示例查询：



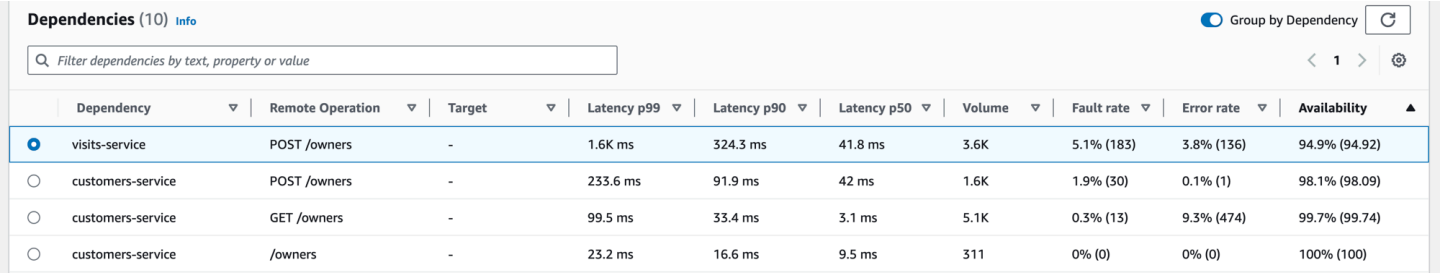
在上图中，CloudWatch 已自动检测与所选点关联的日志组，并将其包含在生成的查询中。

查看您的服务依赖项

选择依赖项选项卡，显示依赖项表以及所有服务操作或单个操作的依赖项的一组指标。该表包含 Application Signals 发现的依赖项列表，其中包括延迟、调用量、故障率、错误率和可用性指标。

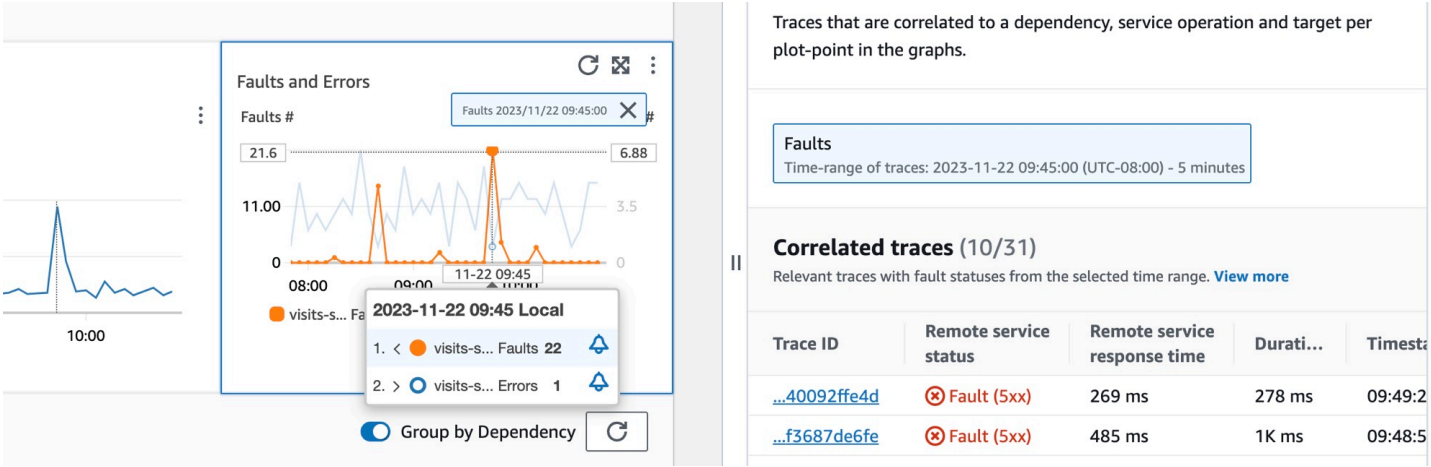
在页面顶部，从向下剪头列表中选择操作查看其依赖项，或者选择全部查看所有操作的依赖项。

通过从筛选条件文本框中选择一个或多个属性来筛选表格，可便于查找所需内容。选择各个属性时，您将在系统引导下选择筛选条件，并在筛选条件文本框下看到完整的筛选条件。随时选择清除筛选条件以移除表格筛选条件。选择表格右上角的按依赖项分组，按服务和操作名称对依赖项进行分组。开启分组后，使用依赖项名称旁边的 + 图标展开或折叠一组依赖项。



依赖项列显示了依赖项服务名称，而远程操作列显示了服务操作名称。调用 AWS 服务时，目标列会显示 DynamoDB 表和 Amazon SNS 队列等 AWS 资源。

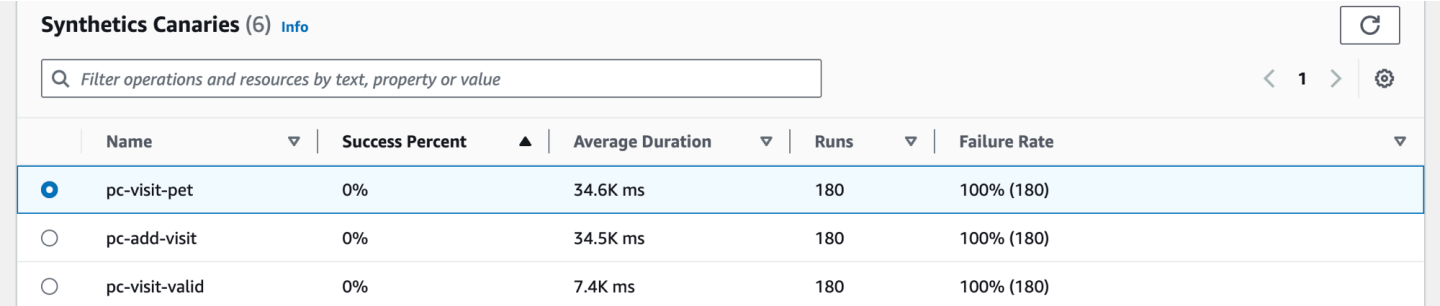
要选择依赖项，请选择依赖项表格中依赖项旁边的选项。此操作将显示一组图表，其中显示调用量、可用性、故障和错误的详细指标。将鼠标悬停在图表中的某个点上，即可查看包含更多信息的弹出窗口。选择图表中的一个点即可打开诊断窗格，其中显示图表中选定点的相关跟踪。从相关跟踪表中选择一个跟踪 ID，打开所选跟踪的 X-Ray 跟踪详细信息页面。



查看您的 Synthetics Canary

选择 Synthetics Canary 选项卡，可显示 Synthetics Canary 表格以及表格中每个 Canary 的一组指标。该表包括成功百分比、平均持续时间、运行次数和失败率的指标。仅显示已为 AWS X-Ray 跟踪启用的 Canary。

使用 Synthetics Canary 表中的筛选条件文本框查找您感兴趣的 Canary。您创建的每个筛选条件都在筛选条件文本框下显示。随时选择清除筛选条件以移除表格筛选条件。



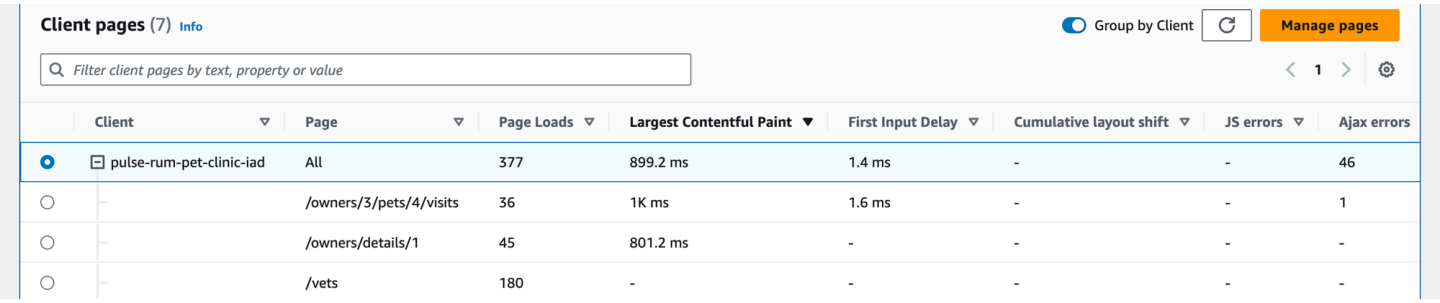
选择 Canary 名称旁边的单选按钮可查看一组选项卡，其中包含图表详细指标，包括成功百分比、错误和持续时间。将鼠标悬停在图表中的某个点上，即可查看包含更多信息的弹出窗口。选择图表中的一个点即可打开诊断窗格，该窗格显示了与选定点相关的 Canary 运行。选择 Canary 运行并选择运行时间以查看所选 Canary 运行的构件，包括日志、HTTP 存档（HAR）文件、屏幕截图以及帮助您排查问题的建议步骤。选择了解更多，以打开 Canary 运行旁边的 CloudWatch Synthetics Canary 页面。

查看您的客户端页面

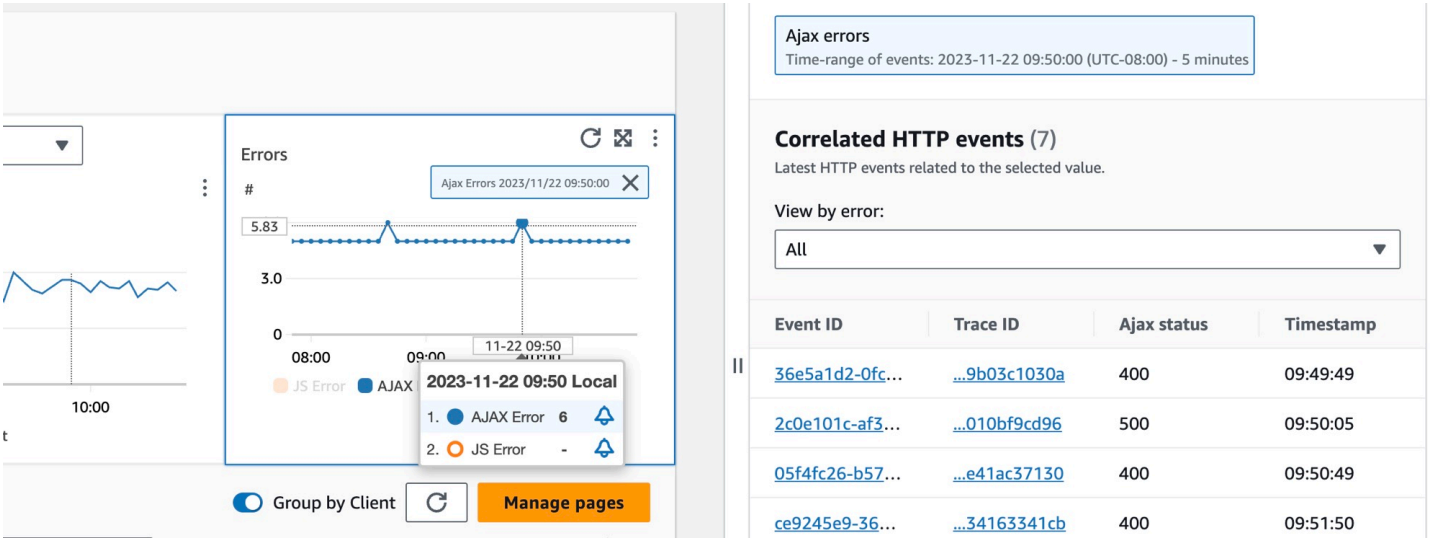
选择客户端页面选项卡以显示调用服务的客户端网页列表。使用选定客户端页面的一组指标来衡量客户在与服务或应用程序交互时的体验质量。这些指标包括页面加载量、Web 重要信息和错误。

要在此表格中显示您的客户端页面，您必须将您的 CloudWatch RUM Web 客户端配置为 X-Ray 跟踪，并开启客户端页面的 Application Signals 指标。选择管理页面，以选择为 Application Signals 指标启用的页面。

使用筛选器文本框在筛选器文本框下方查找您感兴趣的客户端页面或应用程序监视器。选择清除筛选条件以移除表格筛选条件。选择按客户端分组，可按客户端对客户端页面进行分组。分组后，选择客户端名称旁边的 + 图标，即可展开该行并查看该客户端的所有页面。



要选择客户端页面，请在客户端页面表格中选择客户端页面旁边的选项。您将看到一组显示详细指标的图表。将鼠标悬停在图表中的某个点上，即可查看包含更多信息的弹出窗口。选择图表中的一个点即可打开诊断窗格，该窗格显示了图表中选定点的相关性能导航事件。从导航事件列表中选择一个事件 ID，打开所选事件的 CloudWatch RUM 页面视图。



> **Note**
>
> 要查看客户端页面中的 AJAX 错误，请使用 CloudWatch RUM Web 客户端 1.15 版本或更高版本。
>
> 目前，每项服务最多可以显示 100 个操作、Canary 和客户端页面，以及最多 250 个依赖项。

使用 CloudWatch 服务地图查看您的应用程序拓扑并监控运行状况

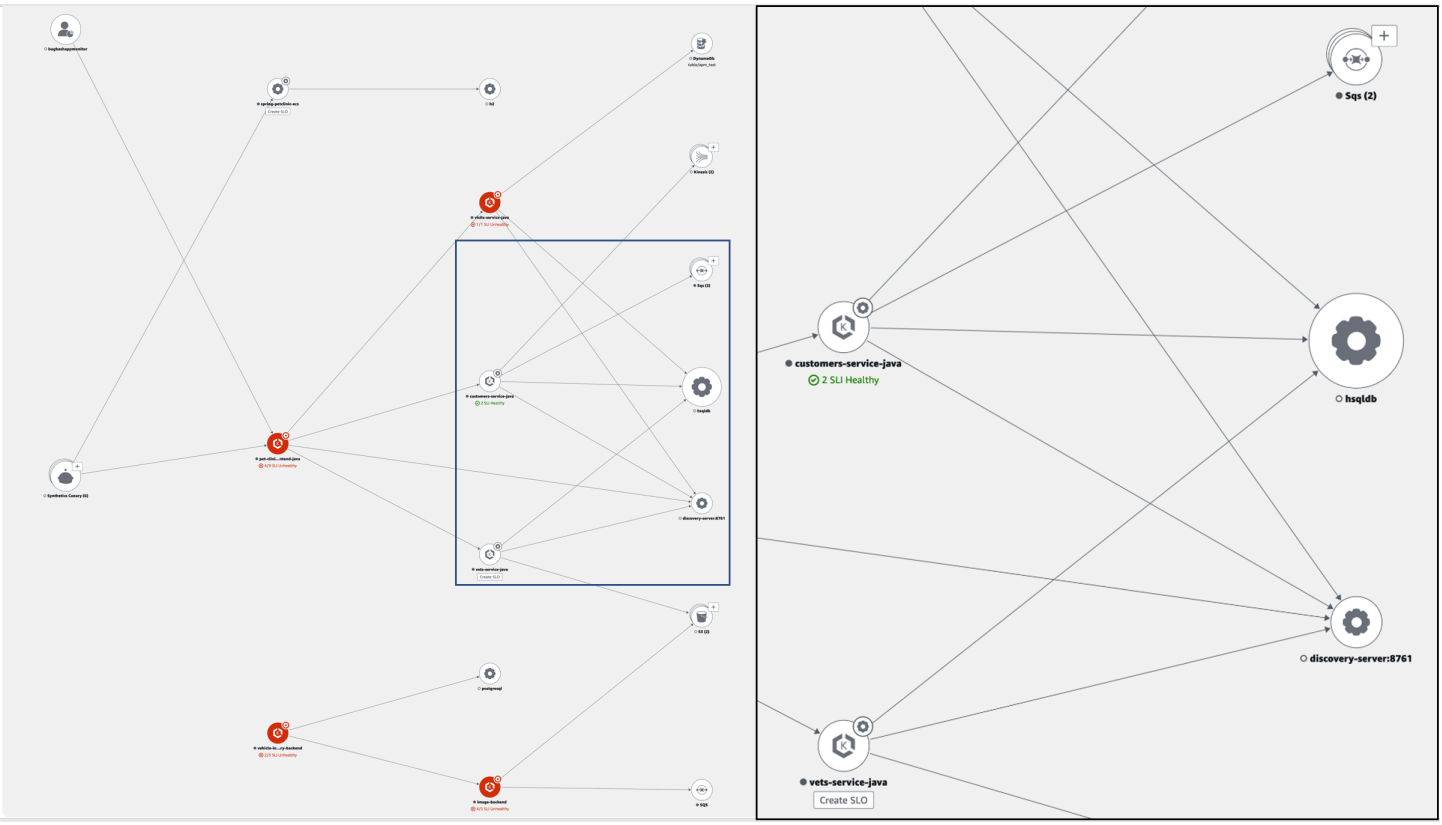
> **Note**
>
> CloudWatch 服务地图取代了 ServiceLens 地图。要按 AWS X-Ray 跟踪查看应用程序地图，请打开 X-Ray 跟踪地图。在 CloudWatch 控制台中，在左侧导航窗格的 X-Ray 部分下选择跟踪地图。

使用服务地图查看应用程序客户端拓扑、Synthetics Canary、服务和依赖项，并监控运行状况。要查看服务地图，请打开 CloudWatch 控制台，然后选择左侧导航窗格的 Application Signals 部分下的服务地图。

在为应用程序启用 Application Signals 后，使用服务地图可以更轻松地监控应用程序的运行状况：

- 查看客户端、Canary、服务和依赖项节点之间的连接，以便了解应用程序拓扑和执行流程。如果服务运营商不是您自己的开发团队，这一点尤其有用。
- 查看哪些服务达到或未达到您的服务级别目标（SLO）。如果某项服务未达到 SLO，则您可以快速确定是否有某个下游服务或依赖项导致此问题或影响多个上游服务。
- 选择单个客户端、Synthetics Canary、服务或依赖项节点以查看相关指标。服务详细信息页面显示有关操作、依赖项、Synthetics Canary 和客户端页面的更多详细信息。
- 筛选和缩放服务地图，从而更便于集中查看应用程序拓扑的某一部分，或者查看整个地图。通过从筛选条件文本框中选择一个或多个属性来创建筛选条件。选择各个属性时，系统将引导您选择筛选条件。您将在筛选条件文本框下看到完整的筛选条件。随时选择清除筛选条件以移除筛选条件。

以下示例服务地图显示了带有边缘的服务，这些边缘将它们连接到与之交互的组件。如果定义了 SLO，则服务地图还会显示运行状况。

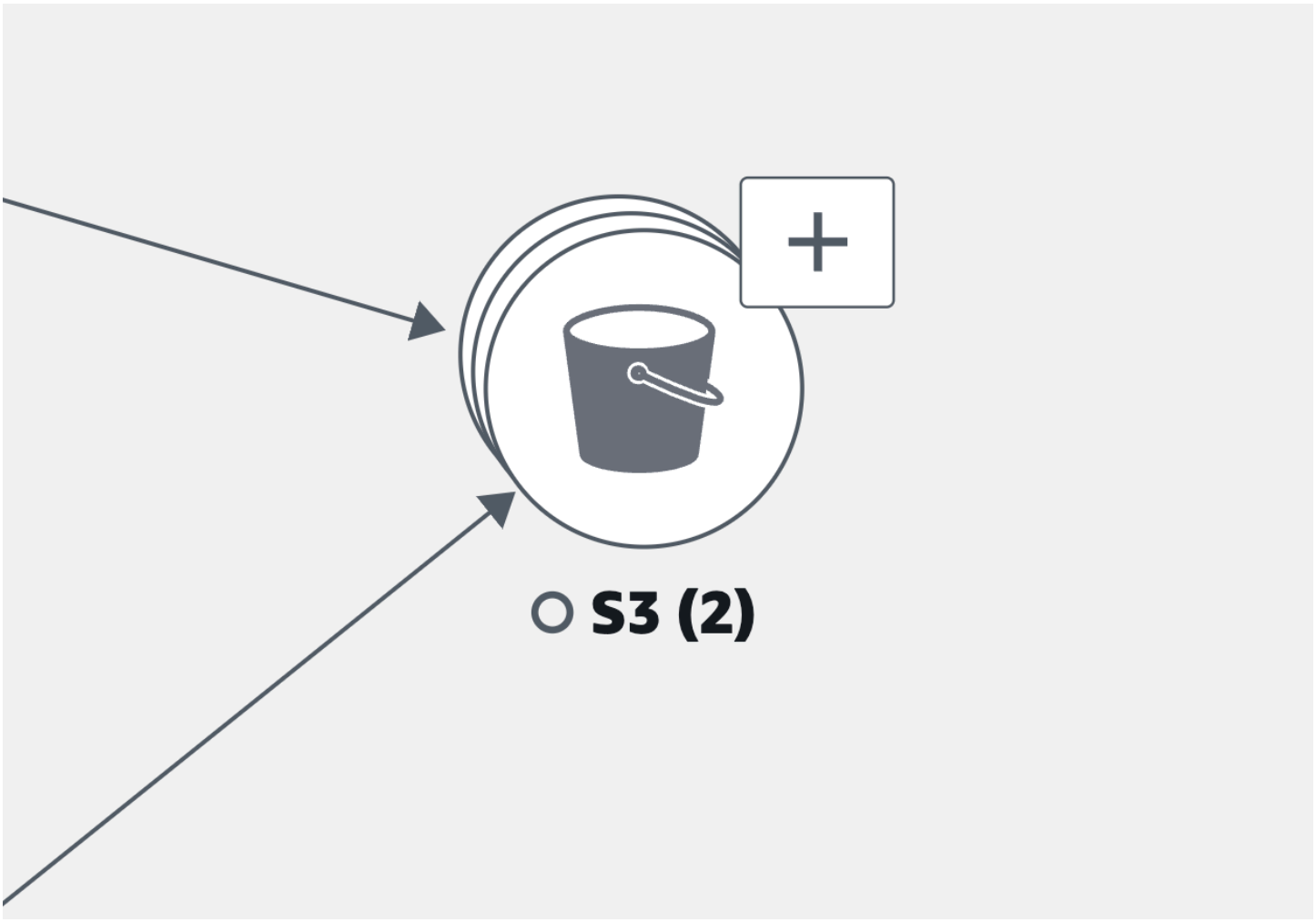


浏览服务地图

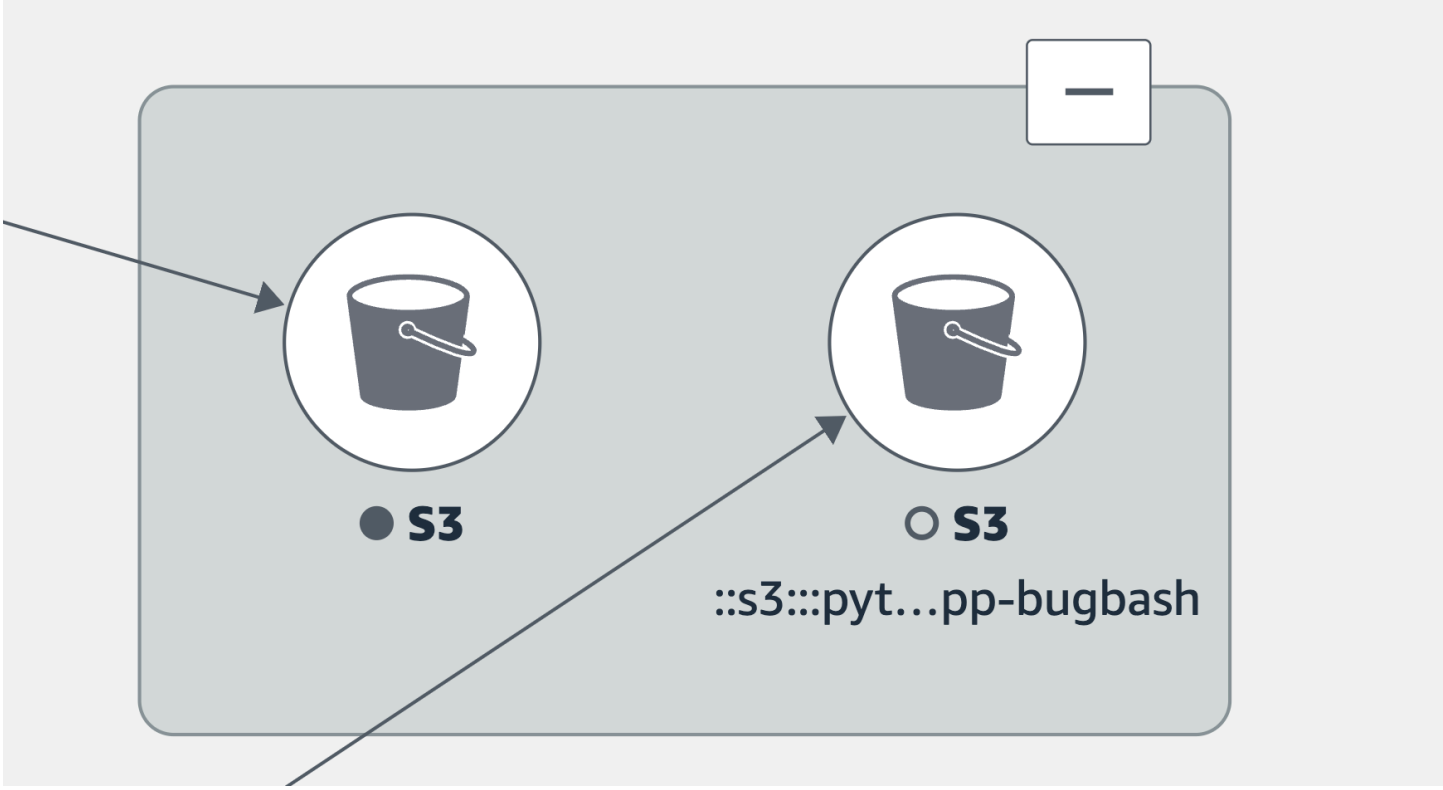
为应用程序启用 Application Signals 后，服务地图将显示代表您的服务和依赖项的节点。

为 CloudWatch RUM 客户端和 Synthetics Canary 开启主动跟踪，即可在地图上查看客户端和 Canary 节点。

默认情况下，同类的 Canary、RUM 客户端和 AWS 服务依赖项在服务地图中组合成一个可扩展的图标。默认情况下，AWS 外的服务依赖项不会组合在一起。例如，在下图中，所有的 Amazon S3 存储桶都组合在一个可扩展图标下：



在上图中，Amazon S3 分组和发端服务之间的标签在依赖项图标下方的括号中显示该组的边缘数量。选择（+）图标展开组并查看其各个元素，如下图所示：

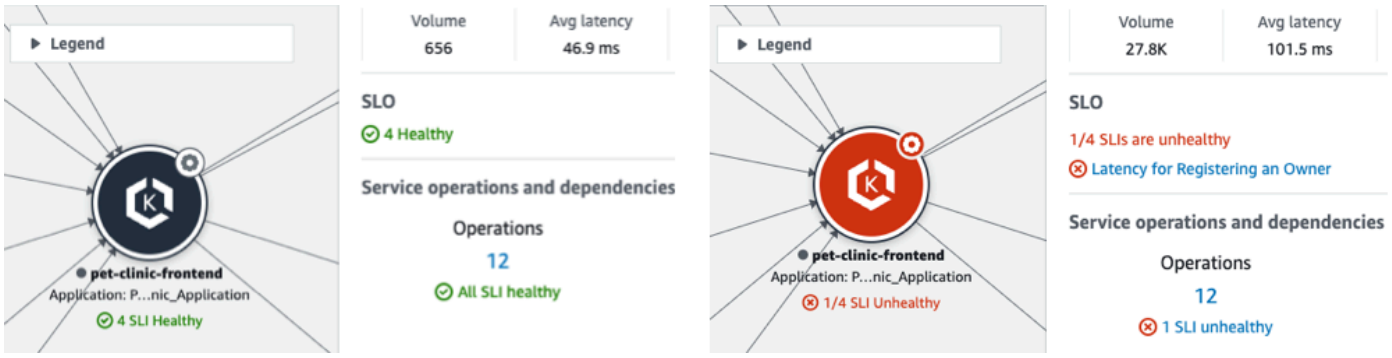


选择选项卡，了解有关浏览每种节点及其之间的边缘（连接）的信息。

View your application services

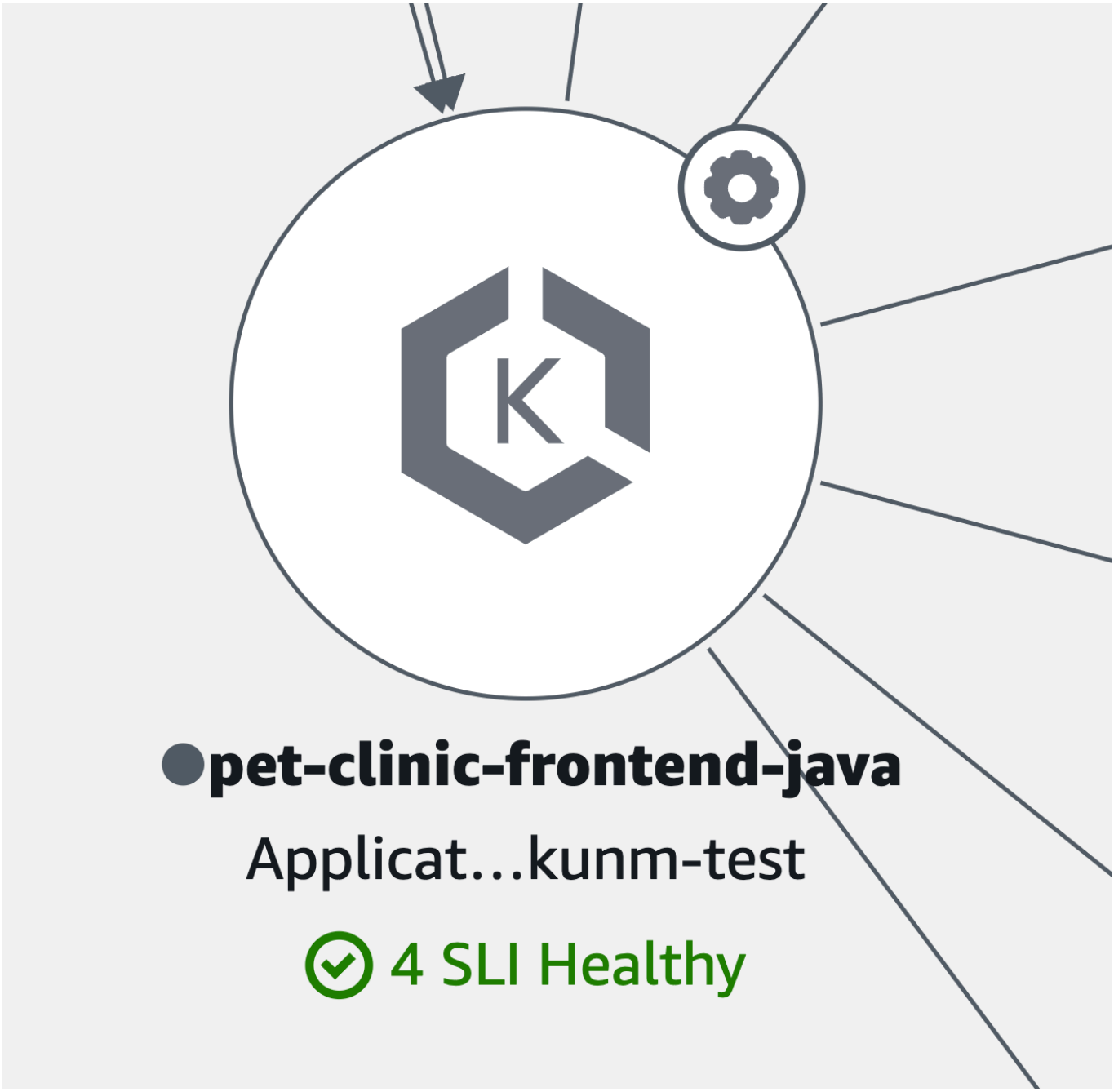
您可以在服务地图中查看您的应用程序服务，以及其 SLO 和服务级别指标（SLI）的状态。如果没有为服务创建 SLO，请选择服务节点下方的创建 SLO 按钮。

服务地图显示您的所有服务。它还显示使用服务的客户和 Canary 以及您的服务调用的依赖项，如下图所示：



以下图标表示服务地图中的应用程序服务的示例：

- Amazon Elastic Kubernetes Service：



- 一个 Kubernetes 容器：

- Amazon Elastic Compute Cloud（Amazon EC2）：

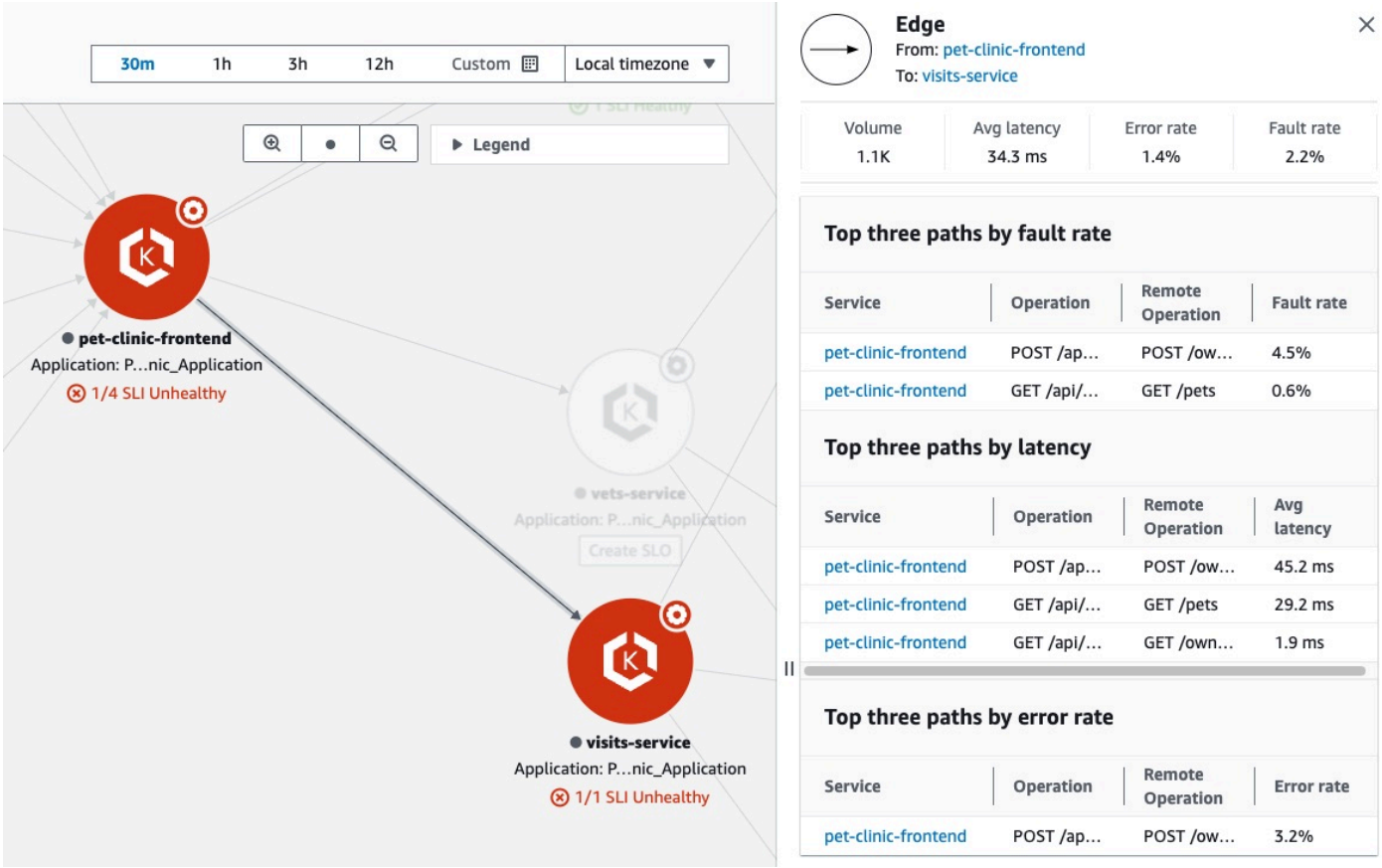
- 以前未列出的其他应用程序服务类型：

选择服务节点后，将开启一个显示详细服务信息的窗格：

- 调用量、延迟、错误和故障率的指标。
- 为 healthy 或 unhealthy 的 SLI 和 SLO 数量。
- 查看有关 SLO 的更多信息的选项。
- 服务操作、依赖项、Synthetics Canary 和客户端页面的数量。
- 选择每个数字以打开其服务详细信息页面的选项。
- 应用程序名称，前提是您已使用 AppRegistry 或 AWS Management Console 主页上的“应用程序”卡将底层计算资源与应用程序相关联。
  - 选择应用程序名称，以在 myApplications 控制台页面中显示应用程序的详细信息。

- 对于托管在 Amazon EKS 中的服务为 `Cluster`、`Namespace` 和 `Workload`，或者托管在 Amazon ECS 或 Amazon EC2 中的服务为 `Environment`。对于 Amazon EKS 托管的服务，请选择任意链接打开 CloudWatch Container Insights。

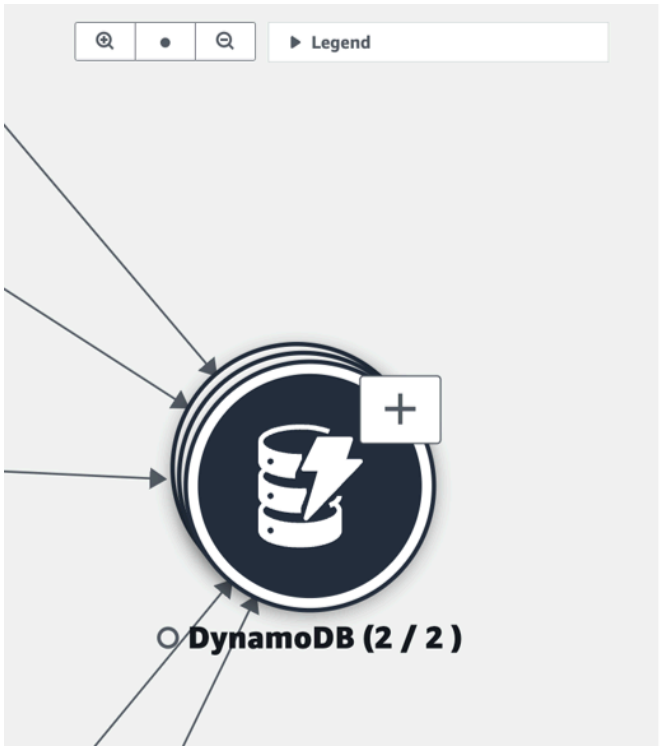
选择服务节点和下游服务或依赖项节点之间的边缘或连接。这将打开一个窗格，其中包含按故障率、延迟和错误率划分的主要路径，如下图中所示。选择窗格中的任何链接，打开服务详细信息页面，查看选定服务或依赖项的详细信息。



View dependencies

应用程序依赖项将显示在服务地图上，并与调用其的服务相连接。

选择依赖项节点以打开窗格，其中包含按故障率、延迟和错误率划分的主要路径。选择任何服务或目标链接，打开服务详细信息页面，查看选定服务或依赖项目标的详细信息，如下面的示例图中所示：



Volume	Avg latency	Error rate	Fault rate
-	-	-	-

Top three paths by fault rate

Service	Remote operation	Fault rate
No paths with faults		

Top three paths by latency

Service	Remote operation	Avg latency
billing-service-ec2-python	PutItem	282.8 ms
billing-service-python	PutItem	75.6 ms
visits-service-java	PutItem	64.9 ms

Top three paths by error rate

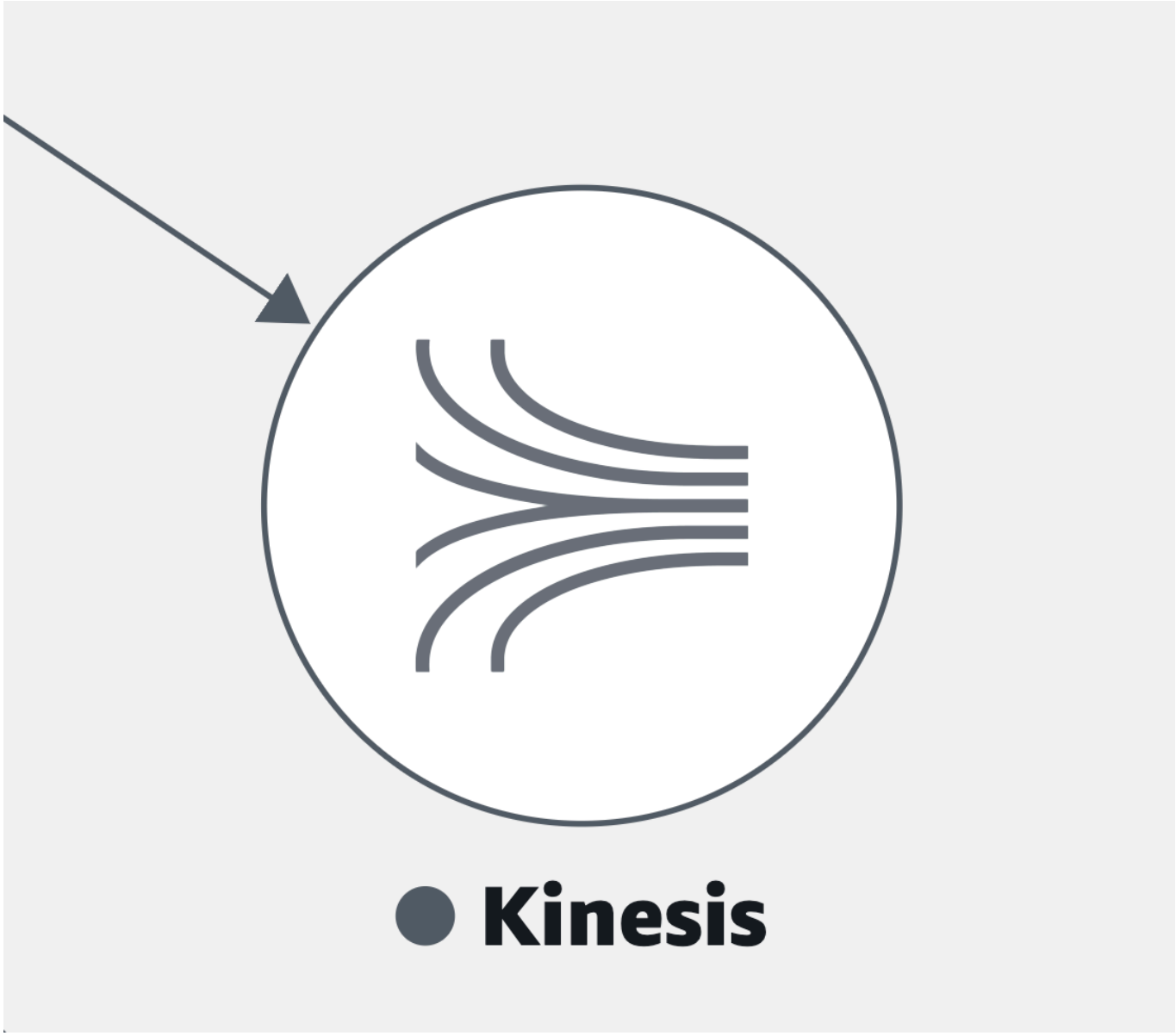
Service	Remote operation	Error rate
visits-service-java	PutItem	9.6%

默认情况下，服务依赖项组合成一个可扩展图标。如上图所示选择（+）图标展开组并查看其各个元素。

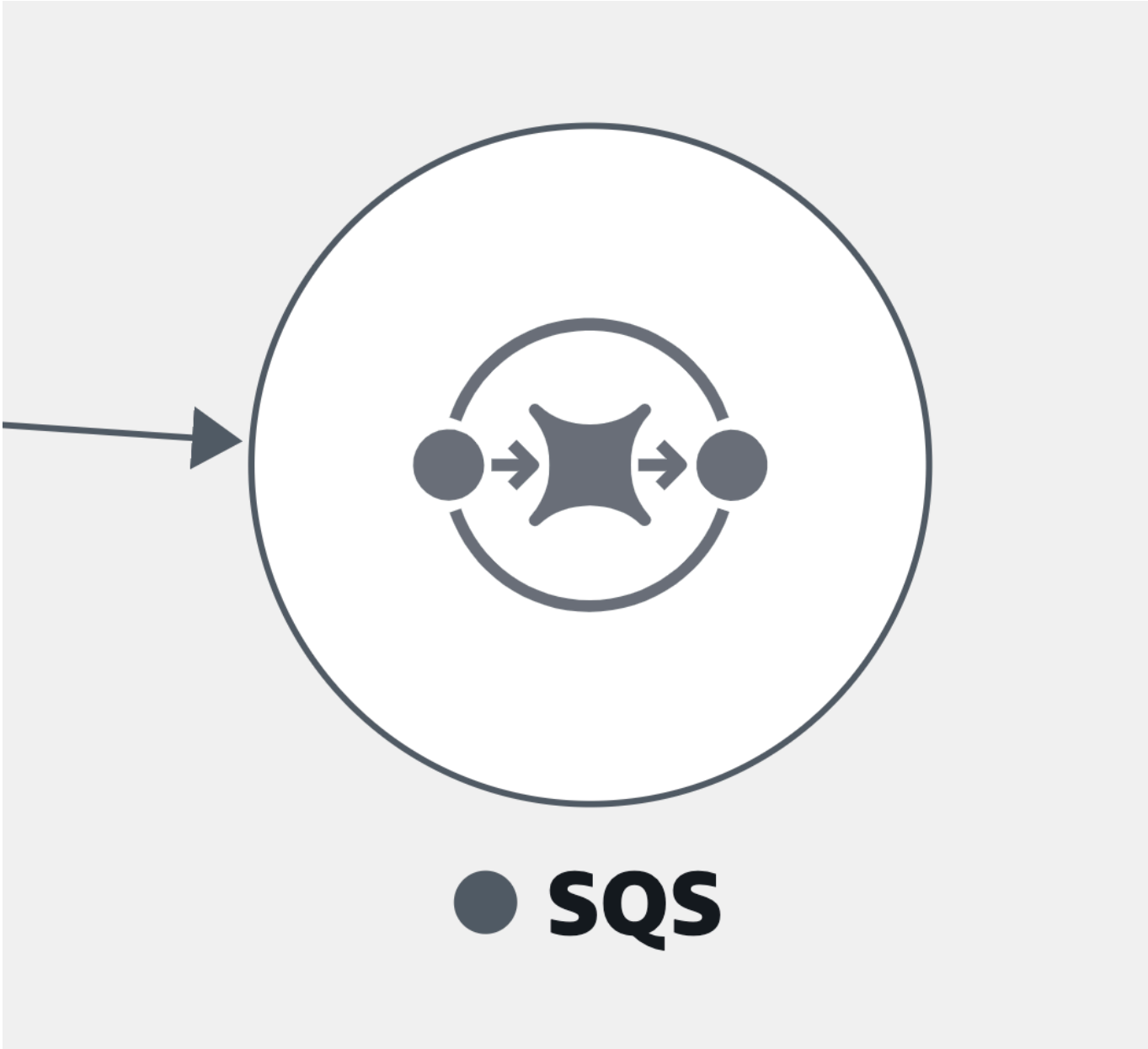
以下图标表示服务地图中依赖项节点的示例：

- Amazon S3 存储桶：

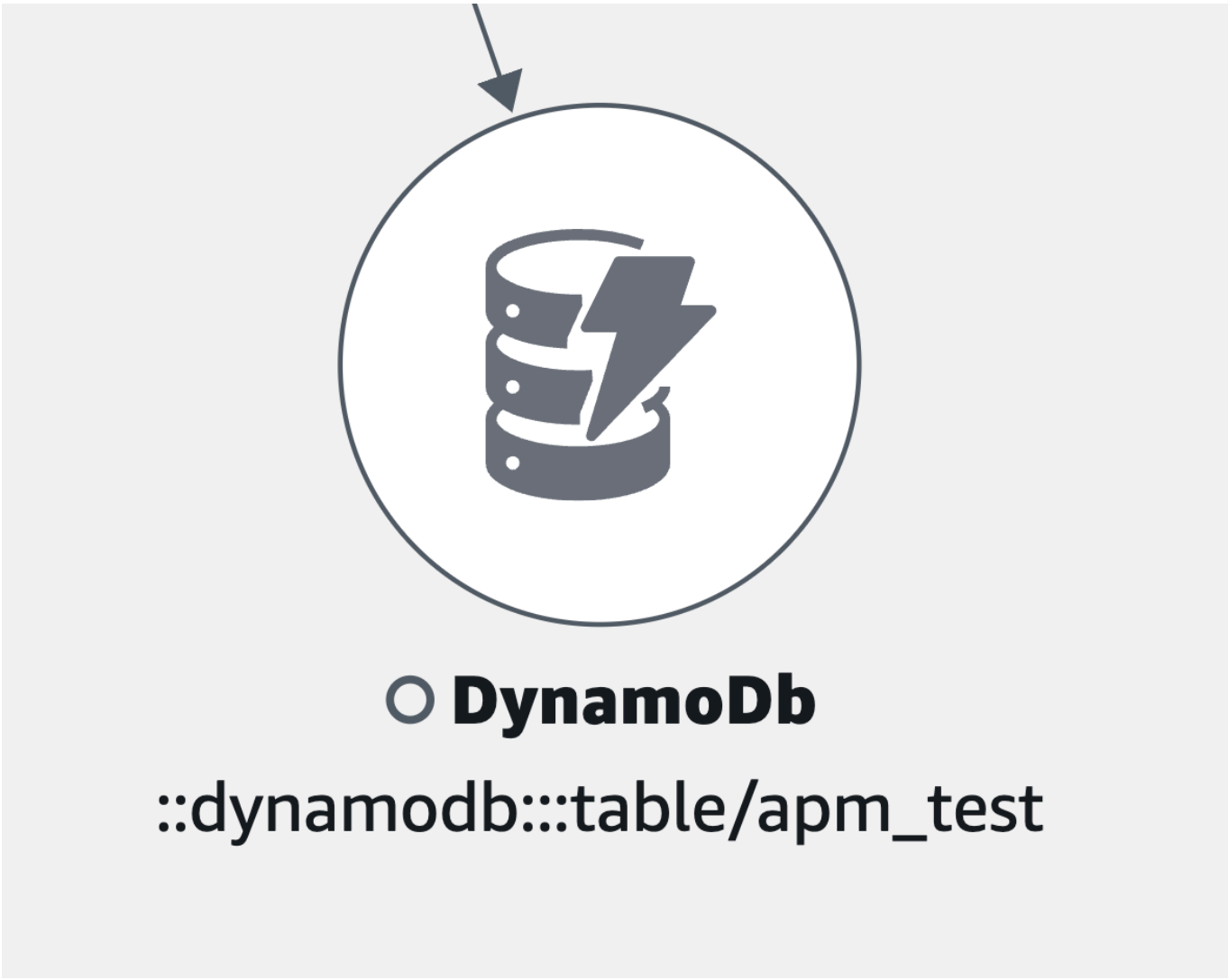
- Amazon Kinesis 流：



- Amazon Simple Queue Service（Amazon SQS）：

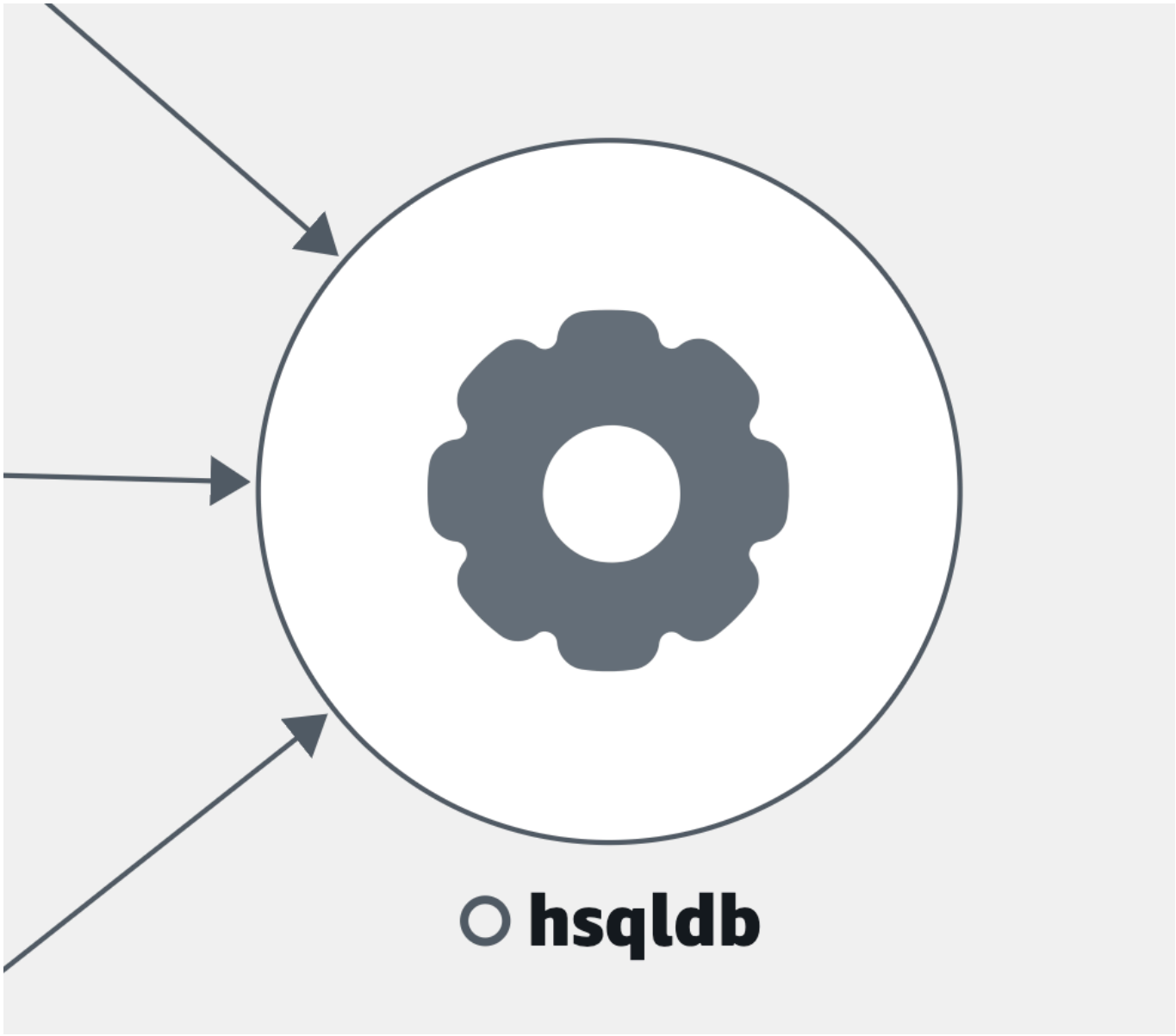


- Amazon DynamoDB 表：



- 一个 Amazon Bedrock 模型：

- 以前未列出的其他依赖项类型：



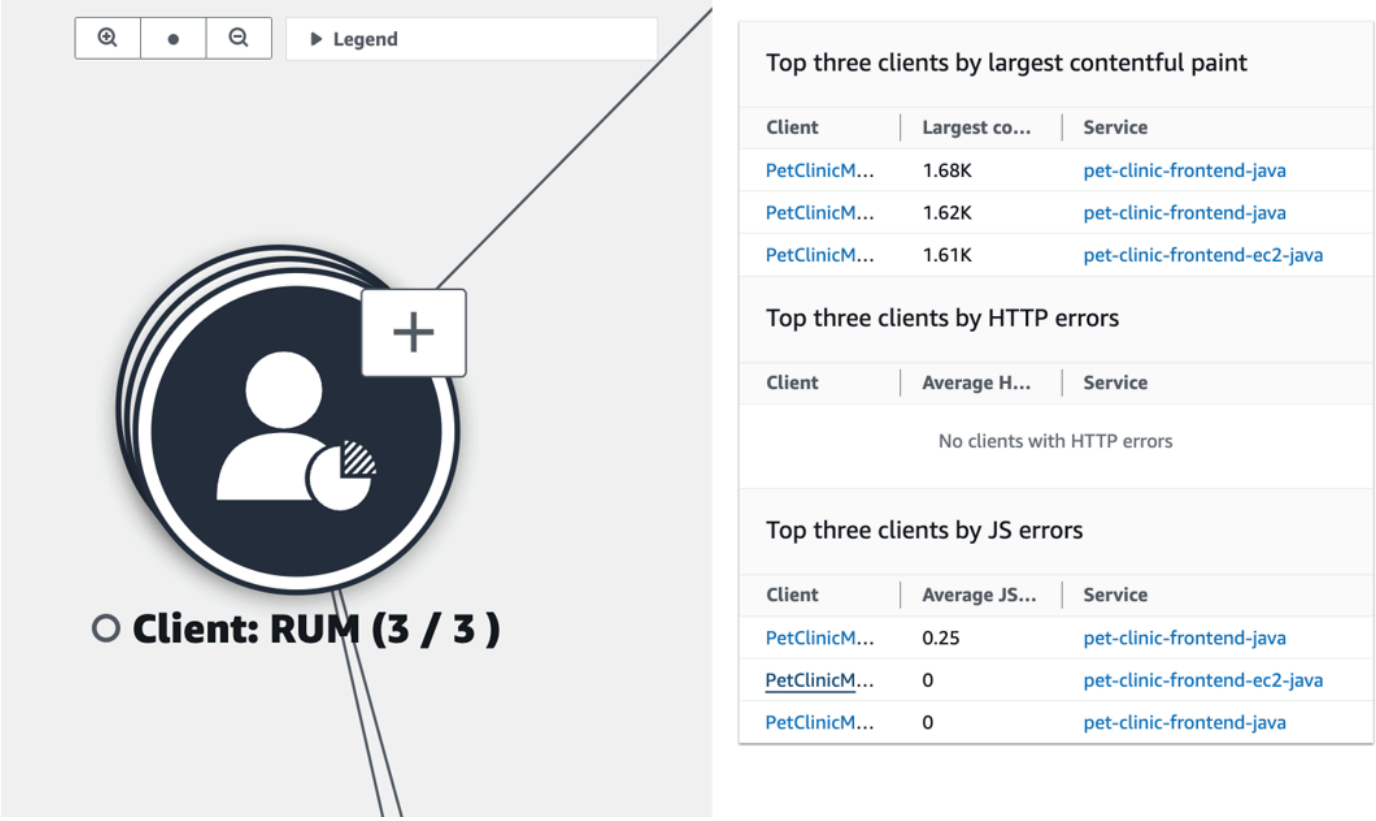
View clients

为 CloudWatch RUM Web 客户端开启 X-Ray 跟踪后，它们会显示在与其调用的服务相连接的服务地图上。

选择客户端节点，打开显示详细客户信息的窗格：

- 页面加载量、平均加载时间、错误和平均 Web 重要信息的指标。
- 显示错误明细的图表。
- 显示 CloudWatch RUM 中客户端详细信息的链接。

默认情况下，RUM 客户端组合成一个可扩展图标。若下图所示选择（+）图标展开组并查看其各个元素。



以下图标代表服务地图中 RUM 客户端的示例：

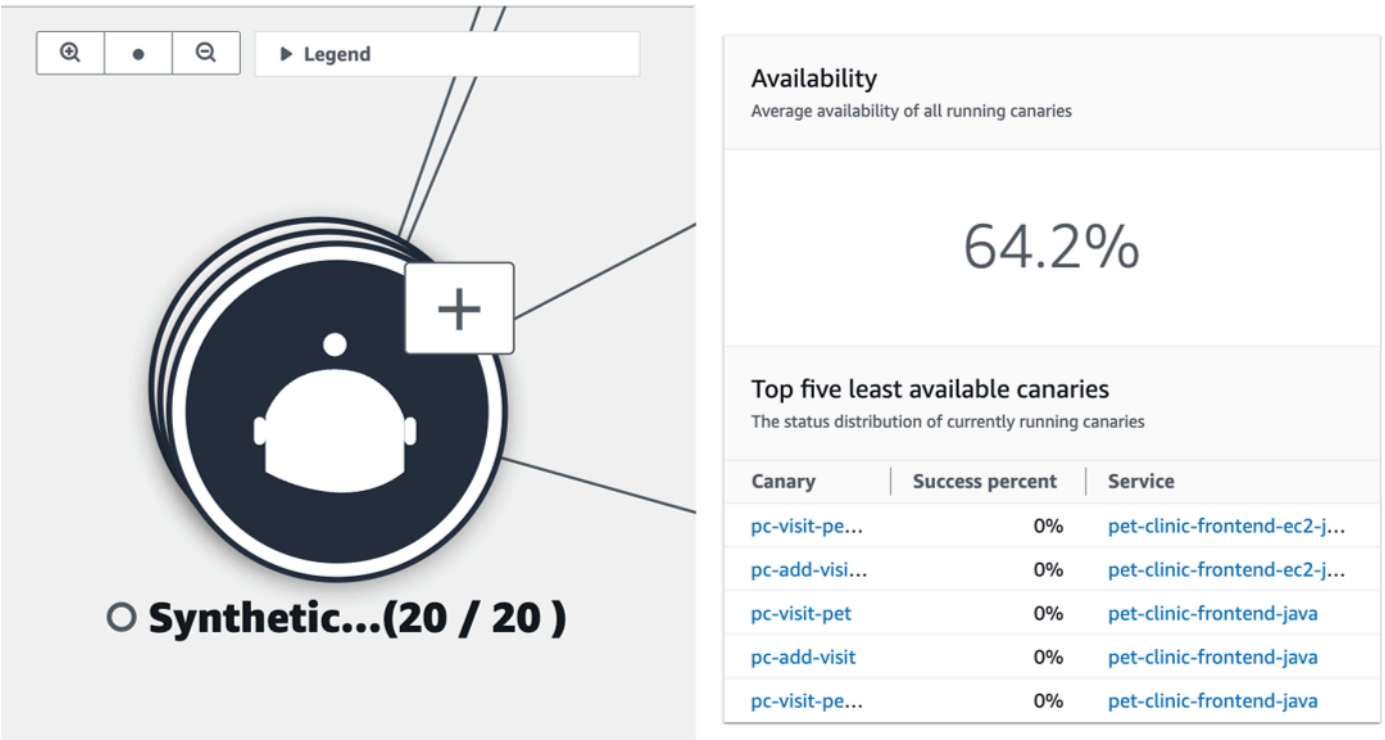
- 一个 RUM 客户端 –

> **Note**
>
> 要查看客户端页面中的 AJAX 错误，请使用 CloudWatch RUM Web 客户端 1.15 版本或更高版本。

View synthetics canaries

为 CloudWatch Synthetics Canary 开启 AWS X-Ray 跟踪后，它们会显示在服务地图上（此服务地图已连接到其调用的服务），如下面的示例图所示：。

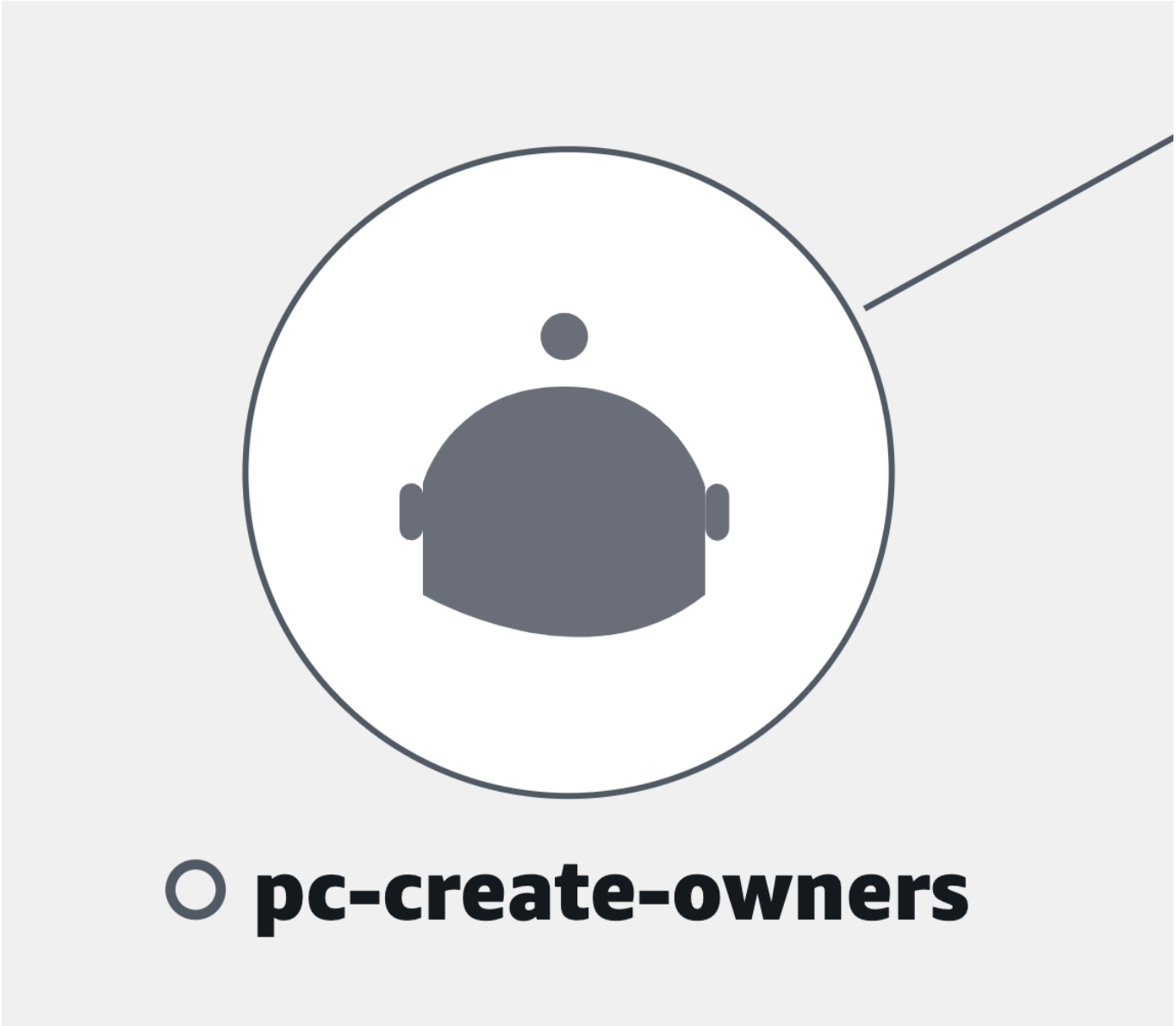
选择 Canary 节点，以打开显示详细 Canary 信息的窗格，如下图中所示：

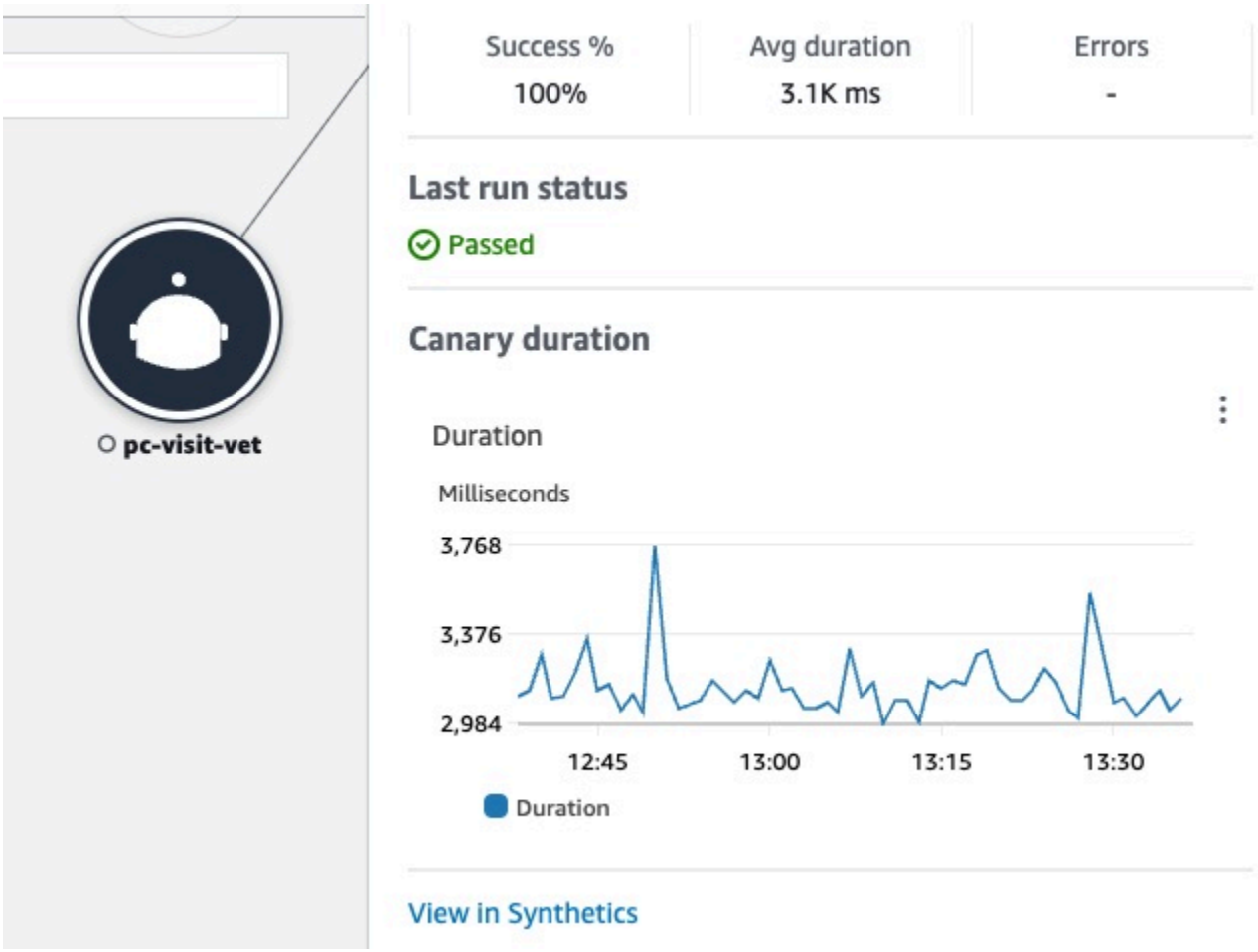


默认情况下，Canary 组合成一个可扩展图标。如上图所示选择（+）图标展开组并查看其各个元素。

以下图标表示服务地图中客户端的示例：

- 一个 Synthetics Canary –


pc-create-owners



在 Canary 节点的窗格中，您可以看到以下内容：

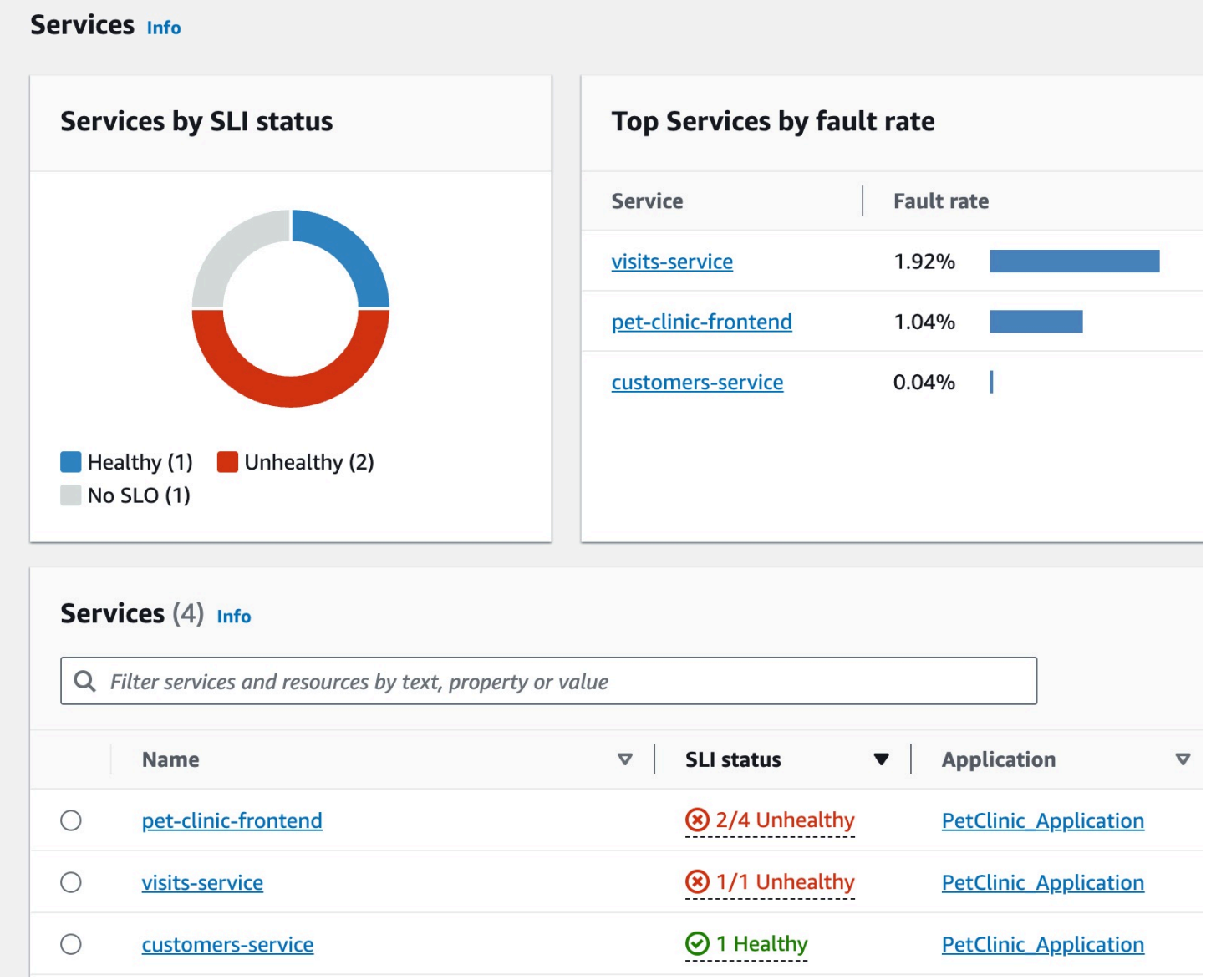
- 成功百分比、平均持续时间和错误的指标。
- 上次 Canary 运行的状态。
- 显示 Canary 运行持续时间的图表。将鼠标悬停在图表序列上，即可查看包含更多信息的弹出窗口。
- 显示 CloudWatch Synthetics 中 Canary 详细信息的链接。

示例：使用 Application Signals 解决运行状况问题

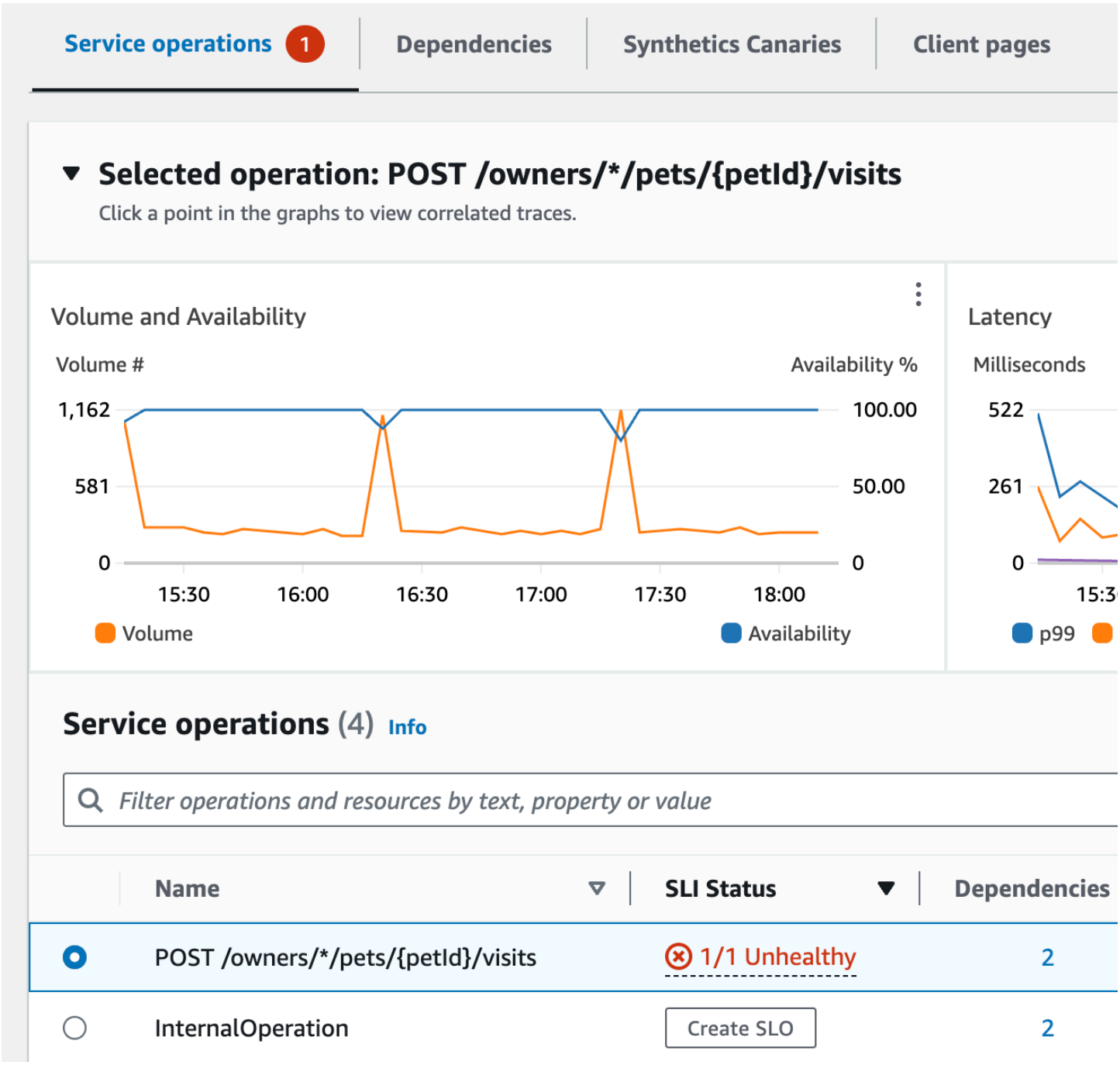
以下场景提供了一个示例，说明如何使用 Application Signals 来监控服务和识别服务质量问题。深入研究，找出潜在根本原因并采取措施解决问题。此示例重点介绍宠物诊所应用程序，该应用程序由多个调用 DynamoDB 等 AWS 服务 的微服务组成。

Jane 是 DevOps 团队的一员，该团队负责监管宠物诊所应用程序的运行状况。Jane 的团队致力于确保该应用程序的高可用性和响应能力。他们使用服务级别目标（SLO）来衡量在这些业务承诺方面的应

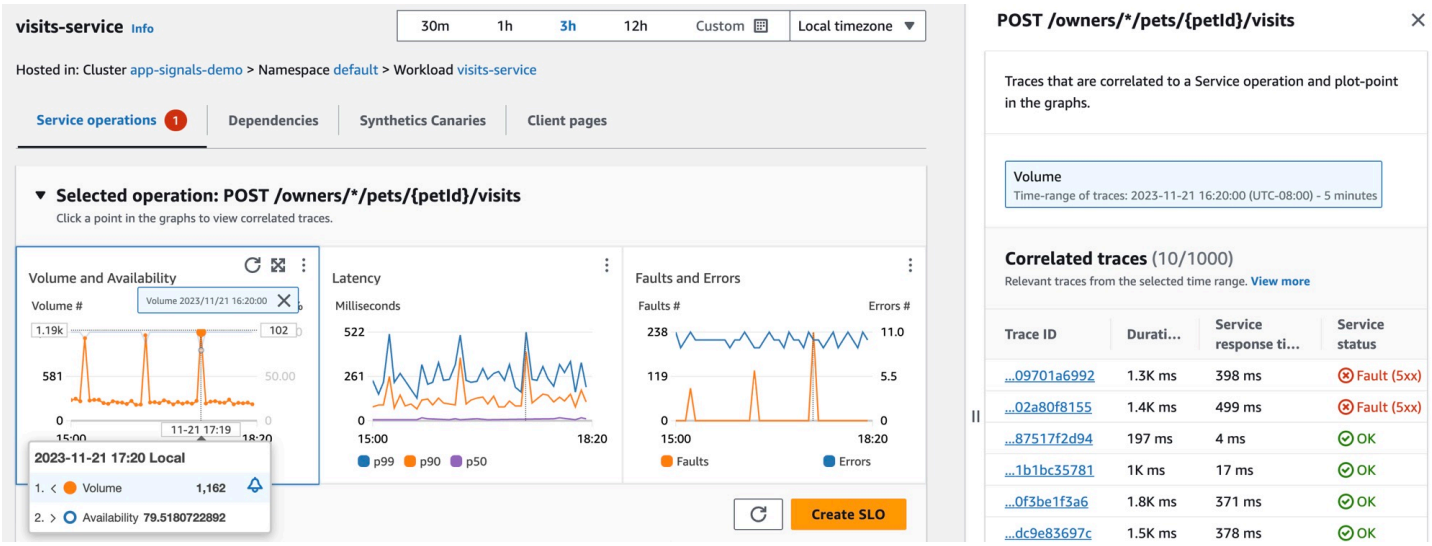
用程序性能。她收到有关多个运行不正常服务级别指标（SLI）的警报。她打开 CloudWatch 控制台并导航到“服务”页面，然后看到有多个服务运行不正常。



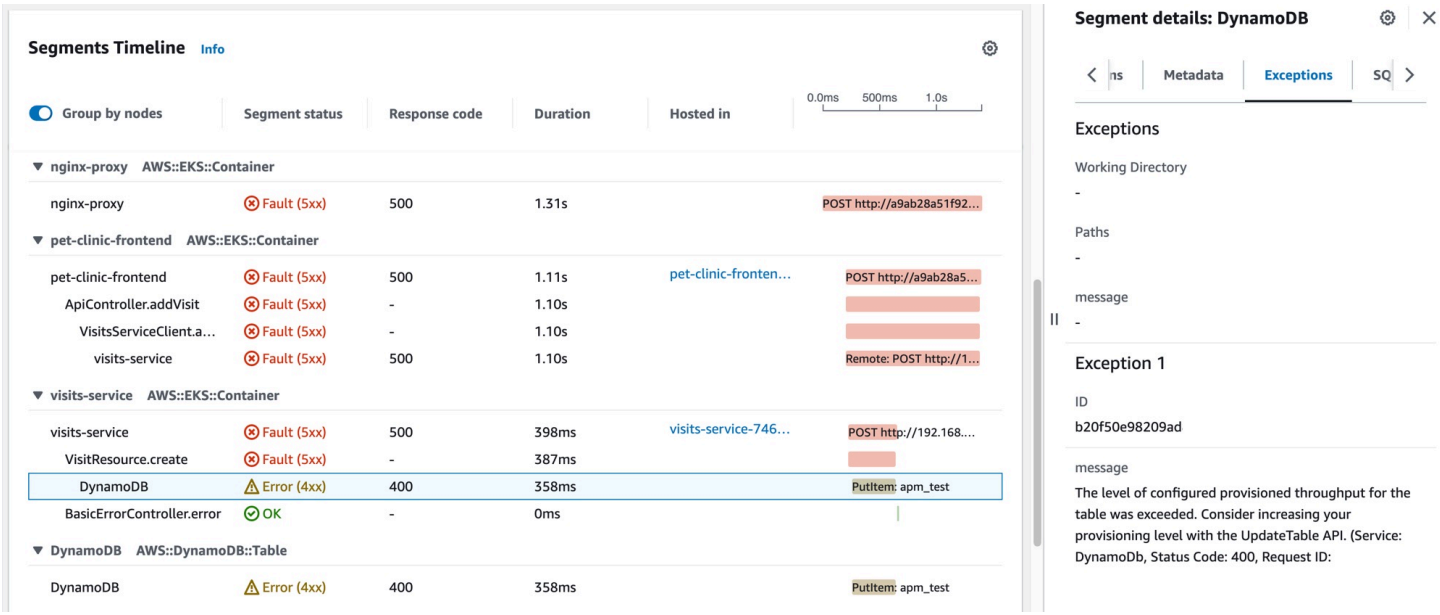
Jane 在该页面顶部看到 `visits-service` 是按故障率排序的顶部服务。她选择了图表中的链接，打开该服务的“服务详细信息”页面。她看到“服务操作”表格中存在不正常的运行状况。她选择了此操作，并在“操作量和可用性”图表中看到，出现的周期性调用量激增似乎与可用性下降有关。



为了更好地观察服务可用性的下降情况，Jane 选择了图表中的一个可用性数据点。这打开了一个抽屉，其中显示与所选数据点相关的 X-Ray 跟踪。她看到有多个跟踪处于故障状态。



Jane 选择了其中一个具有故障状态的关联跟踪，随即打开所选跟踪的 X-Ray 跟踪详细信息页面。Jane 向下滚动到“分段时间线”部分并跟踪调用路径，直到看到对 DynamoDB 表的调用开始返回错误。她选择了 DynamoDB 分段，然后导航到右侧抽屉的“异常”选项卡。



Jane 发现 DynamoDB 资源配置错误，导致在客户端请求激增期间出现错误。DynamoDB 表的预置吞吐量水平周期性超标，导致服务可用性问题以及 SLI 运行不正常。根据这些信息，她的团队得以配置更高水平的预置吞吐量，并确保应用程序的高可用性。

示例：使用 Application Signals 排查与 Amazon Bedrock 模型交互的生成式人工智能应用程序问题

您可以使用 Application Signals 排查与 Amazon Bedrock 模型交互的生成式人工智能应用程序问题。您可以使用 Application Signals 深入探索应用程序及其依赖项（例如 Amazon Bedrock 模型）的指标、跟踪和日志，从而更深入地了解不同模型中的故障（例如模型验证异常、模型配置和延迟等）如何影响最终用户体验。有关更多信息，请参阅 AWS 博客文章 Improve Amazon Bedrock Observability with Amazon CloudWatch Application Signals。

收集的标准应用程序指标

Application Signals 从其发现的服务中收集标准应用程序指标。这些指标与服务性能中最关键的方面有关：延迟、故障和错误。这些指标可以帮助您识别问题、监控性能趋势并优化资源以改善整体用户体验。

下表列出了 Application Signals 收集的指标。这些指标被发送到 `ApplicationSignals` 命名空间中的 CloudWatch。

指标	描述
`Latency`	请求发出后、数据传输开始前的延迟。 单位：毫秒
`Faults`	HTTP 5XX 服务器端故障和 OpenTelemetry 跨度状态错误的计数。 单位：无
`Errors`	HTTP 4XX 客户端错误的计数。这些错误被视为不是由服务问题引起的请求错误。因此，Application Signals 控制面板上显示的 `Availability` 指标不会将这些错误视为服务故障。 单位：无

Application Signals 控制面板上显示的 `Availability` 指标的计算公式为 （1 - **`Faults`**/总数）*100。回复总数包括所有回复，并源自 `SampleCount(Latency)`。成功响应是指没有 5XX 错误的所有响应。当 Application Signals 计算 `Availability` 时，4XX 响应会被视为成功响应。

收集的维度和维度组合

为每个标准应用程序指标定义了以下维度。有关维度的更多信息，请参阅 尺寸。

为服务指标和依赖项指标收集不同的维度。在 Application Signals 发现的服务中，当微服务 A 调用微服务 B 时，微服务 B 正在处理请求。在这种情况下，微服务 A 发出依赖项指标，微服务 B 发出服务指标。当客户端调用微服务 A 时，微服务 A 正在处理请求并发出服务指标。

服务指标的维度

为服务指标收集以下维度。

维度	描述
`Service`	服务的名称。 最大值为 255 个字符。
`Operation`	API 操作或其他活动的名称。 最大值为 1024 个字符。当前，仅当操作名称不超过 194 个字符时，您才能为操作设置服务级别目标。
`Environment`	运行服务的环境的名称。如果服务未在 Amazon EKS 上运行，则可以在 `OTEL_ATTRIBUTE_RESOURCES` 参数中为 `deployment.environment` 指定可选的自定义值。 最大值为 259 个字符。

当您在 CloudWatch 控制台中查看这些指标时，可以使用以下维度组合进行查看：

- `[Environment, Service, Operation, [Latency, Error, Fault]]`
- `[Environment, Service, [Latency, Error, Fault]]`

依赖项指标的维度

为依赖项指标收集以下维度：

维度	描述
Service	服务的名称。 最大值为 255 个字符。
Operation	API 操作或其他操作的名称。 最大值为 1024 个字符。
RemoteService	所调用远程服务的名称。 最大值为 255 个字符。
RemoteOperation	所调用 API 操作的名称。 最大值为 1024 个字符。
Environment	运行服务的环境的名称。如果服务未在 Amazon EKS 上运行，则可以在 OTEL_ATTRIBUTE_RESOURCES 参数中为 deployment.environment 指定可选的自定义值。 最大值为 259 个字符。
RemoteEnvironment	运行依赖项服务的环境的名称。当服务调用依赖项并且它们都在同一集群中运行时，会自动生成 RemoteEnvironment 参数。否则，不会生成 RemoteEnvironment ，也不会在服务依赖项的指标中报告该参数。目前仅在 Amazon EKS 和 K8S 平台上可用。 最大值为 259 个字符。
RemoteResourceIdentifier	远程调用所调用资源的名称。如果服务调用远程 AWS 服务，则会自动生成 RemoteResourceIdentifier 参数。否则，不会生成 RemoteResourceIdentifier ，也不会在服务依赖项的指标中报告该参数。 最大值为 1024 个字符。
RemoteResourceType	远程调用所调用资源的类型。仅在定义 RemoteResourceIdentifier 时需要。

维度	描述
	最大值为 1024 个字符。

当您在 CloudWatch 控制台中查看这些指标时，可以使用以下维度组合进行查看：

在 Amazon EKS 集群上运行

- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResoureIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, [Latency, Error, Fault]]
- [RemoteService]
- [RemoteService, RemoteResourceIdentifier, RemoteResourceType]

服务级别目标（SLO）

您可以使用 Application Signals 为关键业务运营的服务创建服务级别目标。创建这些服务的 SLO 后，您即可在 SLO 控制面板上对其进行跟踪，一目了然地了解自己最重要的运营。

除了创建快速视图供运营商查看关键运营的当前状态，您还可以使用 SLO 跟踪服务的长期性能，从而确保其符合您的预期。如果您与客户签订了服务级别协议，SLO 则是确保符合这些协议的绝佳工具。

使用 SLO 评估服务的运行状况，首先要根据关键性能指标（服务级别指标（SLI））设定清晰且可衡量的目标。SLO 会根据您设置的阈值和目标跟踪 SLI 性能，并报告您的应用程序性能与阈值的差异。

Application Signals 帮助您设置关键性能指标的 SLO。Application Signals 自动收集其发现的各项服务和操作的 `Latency` 和 `Availability` 指标，这些指标通常很适合用作 SLI。您可以通过 SLO 创建向导将这些指标用于 SLO。然后，您可以使用 Application Signals 控制面板跟踪所有 SLO 的状态。

您可以为您的服务调用或使用的特定操作设置 SLO。除了使用 `Latency` 和 `Availability` 指标外，您还可以将任何 CloudWatch 指标或指标表达式用作 SLI。

创建 SLO 对于充分利用 CloudWatch Application Signals 至关重要。创建 SLO 后，您可以在 Application Signals 控制台中查看其状态，以快速查看哪些关键服务和操作运行正常或不正常。由 SLO 进行跟踪的主要好处如下：

- 您的服务运营商可以更轻松地查看根据 SLI 衡量的关键服务的当前运行状况。然后即可快速对运行不正常的服务和操作进行分类和识别。
- 您可以根据可衡量的业务目标，长期跟踪您的服务性能。

通过选择要设置 SLO 的对象，您可以确定重要内容的优先级。Application Signals 控制面板会自动显示有关您已确定优先级的信息。

在创建 SLO 时，您还可以选择通过同时创建 CloudWatch 警报来监控 SLO。您可以通过设置警报监控超出阈值的情况，也可以设置监控警告级别的警报。如果 SLO 指标超出所设阈值或接近警告阈值，这些警报可以自动通知您。例如，如果 SLO 接近警告阈值，您便可以得知，您的团队可能需要减慢应用程序的流失速度，以确保实现长期性能目标。

主题

- SLO 概念
- 创建 SLO
- 查看 SLO 状态并对其进行分类

- 编辑现有 SLO
- 删除 SLO

SLO 概念

SLO 包括以下组件：

- 服务级别指标（SLI），即您指定的关键性能指标。代表了您的应用程序所需的性能水平。Application Signals 会自动收集其发现的各项服务和操作的关键指标 `Latency` 和 `Availability`，这些指标通常很适合用作 SLI。

  您可以选择用于 SLI 的阈值。例如，延迟时间为 200 毫秒。
- 目标或达标率目标，即预计 SLI 在各个时间间隔内达到阈值的时间的百分比。时间间隔可以短至几小时，也可以长达一年。

  间隔可以是日历间隔，也可以是滚动间隔。
  - 日历间隔与日历一致，例如每月跟踪的 SLO。CloudWatch 会根据一个月的天数自动调整运行状况、预算和达标数量。日历间隔更适合用于按照日历进行衡量的业务目标。
  - 滚动间隔采用滚动计算。滚动间隔更适合用于跟踪应用程序的近期用户体验。
- 周期是较短的时间长度，许多周期构成一个间隔。将应用程序在间隔内各个周期的性能与 SLI 进行比较。在各个周期确定应用程序已达到或未达到必要的性能水平。

例如，日历间隔为一天、周期为 1 分钟的 99% 目标，意味着应用程序必须在当日的所有 1 分钟周期中，有 99% 的周期达到或实现成功阈值。这样才能实现当日的 SLO。次日即为新的评估间隔，应用程序必须在次日有 99% 的 1 分钟周期达到或实现成功阈值才能实现次日的 SLO。

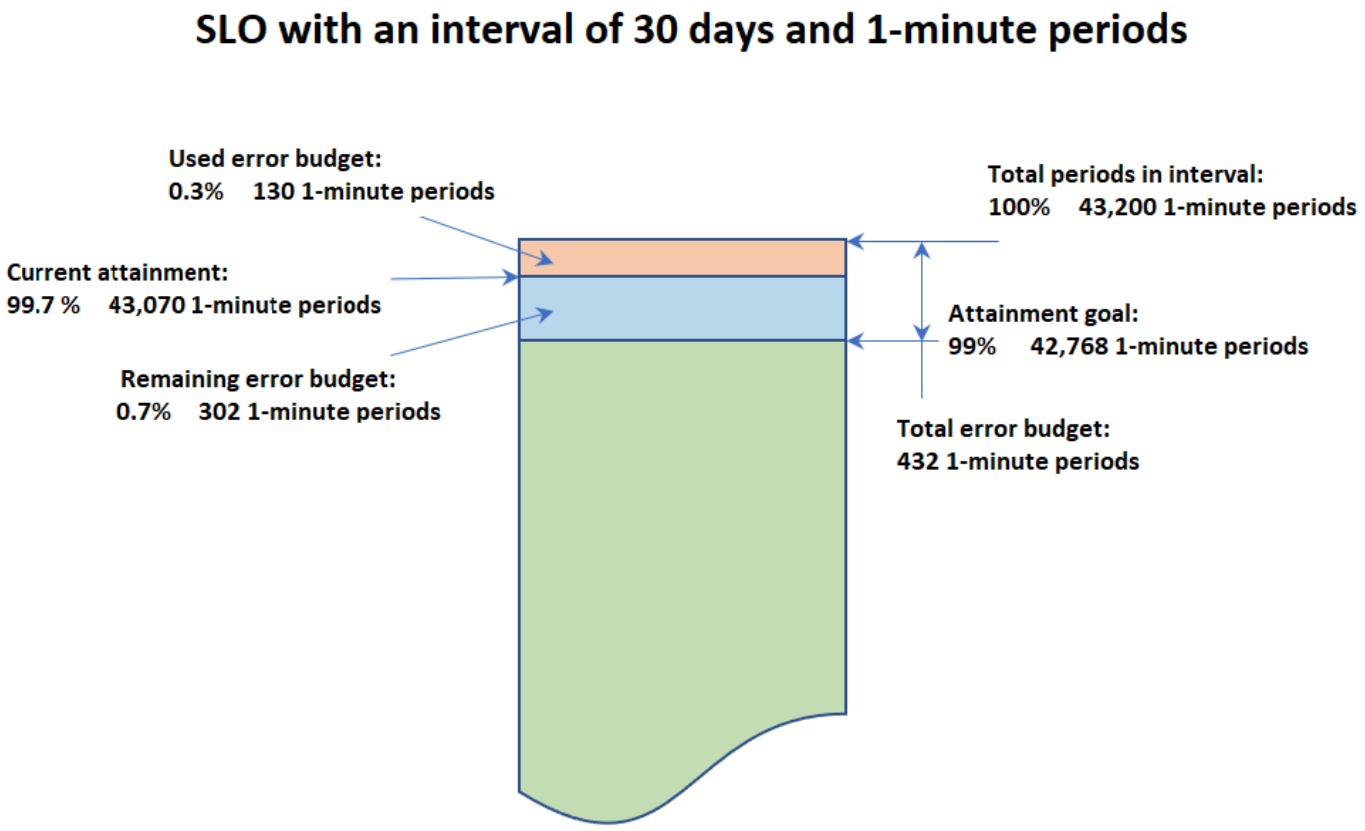
SLI 可以基于 Application Signals 收集的其中一项新标准应用程序指标。或者可以使用任何 CloudWatch 指标或指标表达式。可用于 SLI 的标准应用程序指标为 `Latency` 和 `Availability`。`Availability` 表示成功响应除以请求总数的结果。计算方法为 (1 - 故障率)*100，其中故障响应为 `5xx` 错误。成功响应表示没有 `5XX` 错误的响应。`4XX` 响应视为成功响应。

> **Note**
>
> 目前仅支持基于周期进行计算。未来版本计划支持基于数量或请求进行计算。

计算错误预算和达标率

在查看 SLO 相关信息时，您会看到其当前运行状况和错误预算。错误预算是在间隔内可以超出阈值但仍能达到 SLO 的时间长度。总错误预算是在整个间隔内容许的总超限时间。剩余错误预算是当前间隔内容许的剩余超限时间。这一数据为从总错误预算中扣除已发生超限时间的结果。

下图说明了间隔为 30 天、周期为 1 分钟、达标率目标为 99% 时的达标率和错误预算概念。30 天包括 43200 个 1 分钟周期。43200 的 99% 为 42768，因此当月必须有 42768 分钟处于运行正常状态才能达到 SLO。到目前为止，在当前间隔内，有 130 个运行不正常的 1 分钟周期。



确定每个周期是否成功

在每个周期内，SLI 数据会根据用于 SLI 的统计数据聚合到单个数据点中。此数据点表示周期的整个长度。将单个数据点与 SLI 阈值进行比较，以确定该周期运行是否正常。在控制面板上看到当前时间范围内运行不正常的周期后，您的服务运营商便得知需要对服务进行分类。

如果确定该周期运行不正常，则会根据错误预算将整个周期长度全部计为失败。您可以通过跟踪错误预算，了解服务是否长时间达到所需性能水平。

创建 SLO

我们建议您同时设置关键应用程序的延迟和可用性 SLO。Application Signals 收集的这些指标与共同业务目标一致。

您还可以根据任何 CloudWatch 指标或任何生成单个时间序列的指标数学表达式设置 SLO。

当您首次在账户中创建 SLO 时，CloudWatch 会自动在您的账户中创建 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色（如果该角色尚不存在）。此服务相关角色允许 CloudWatch 收集 CloudWatch Logs 数据、X-Ray 跟踪数据、CloudWatch 指标数据，以及您账户中应用程序的标记数据。有关 CloudWatch 服务相关角色的更多信息，请参阅 为 CloudWatch 使用服务相关角色。

创建 SLO

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务级别目标（SLO）。
3. 请选择创建 SLO。
4. 输入 SLO 的名称。纳入服务或操作的名称以及适当的关键字（例如延迟或可用性）有助于您在分类期间快速识别 SLO 状态所表示的含义。
5. 对于设置服务级别指标（SLI），执行以下任一操作：
   - 为任一标准应用程序指标 `Latency` 或者 `Availability` 设置 SLO：
     a. 选择服务操作。
     b. 选择此 SLO 将监控的服务。
     c. 选择此 SLO 将监控的操作。

        选择服务和选择操作下拉列表中填充了过去 24 小时处于活动状态的服务和操作。
     d. 选择可用性或延迟，然后设置阈值。
   - 为任何 CloudWatch 指标或 CloudWatch 指标数学表达式设置 SLO：
     a. 选择 CloudWatch 指标。
     b. 选择选择 CloudWatch 指标。

        出现选择指标屏幕。使用浏览或查询选项卡查找所需指标或创建指标数学表达式。

        选择所需指标后，选择图表化指标选项卡，然后选择要用于 SLO 的统计数据和周期。然后选择 Select metric（选择指标）。

有关这些屏幕的信息，请参阅 绘制指标图表 和 向 CloudWatch 图表中添加数学表达式。

c. 在设置条件中，选择 SLO 的比较运算符和阈值作为成功指标。

6. 如果您在步骤 5 中选择了服务操作，则可以选择其他设置，然后调整此 SLO 的周期长度。
7. 设置 SLO 的间隔和达标率目标。有关间隔和达标率目标以及二者如何协同工作的更多信息，请参阅 SLO 概念。
8. （可选）为 SLO 设置一个或多个 CloudWatch 警报或警告阈值。

a. 如果根据应用程序的 SLI 性能判断其运行不正常，则 CloudWatch 警报可以使用 Amazon SNS 主动通知您。

要创建警报，请选中任一警报复选框，然后输入或创建 Amazon SNS 主题，以便在警报进入 ALARM 状态时用于发送通知。有关 CloudWatch 警报的更多信息，请参阅 使用 Amazon CloudWatch 告警。创建警报会产生费用。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

b. 如果您设置了警告阈值，其将显示在 Application Signals 屏幕中，从而帮助您识别可能无法实现的 SLO（即使此类 SLO 当前运行正常）。

要设置警告阈值，请在警告阈值中输入阈值。当 SLO 的错误预算低于警告阈值时，多个 Application Signals 屏幕中的 SLO 都会标有警告标记。警告阈值还会显示在错误预算图形中。您也可以创建基于警告阈值的 SLO 警告警报。

9. 要向此 SLO 添加标签，请选择标签选项卡，然后选择添加新标签。标签可帮助您管理、识别、组织、搜索和筛选资源。有关标记的更多信息，请参阅标记您的 AWS 资源。

> **Note**
>
> 如果与此 SLO 相关的应用程序已在 AWS Service Catalog AppRegistry 中注册，则可以使用 `awsApplication` 标签将此 SLO 与 AppRegistry 中的该应用程序关联。有关更多信息，请参阅 What is AppRegistry?

10. 请选择创建 SLO。如果您还选择了创建一个或多个警报，为体现这一点，按钮名称会变更。

查看 SLO 状态并对其进行分类

您可以使用 CloudWatch 控制台中的服务级别目标或服务选项快速查看 SLO 的运行状况。服务视图提供运行不正常的服务比率的概览视图，该比率根据您设置的 SLO 计算得出。有关使用服务选项的更多信息，请参阅 使用 Application Signals 监控应用程序的运行状况。

服务级别目标视图提供了您的组织的宏观视图。您可以整体查看已实现和未实现的 SLO。这有助于您了解，根据所选 SLI，有多少服务和操作的性能长期达到您的预期。

使用服务级别目标视图查看所有 SLO

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务级别目标（SLO）。

   随即显示服务级别目标（SLO）列表。

   您可以在 SLI 状态列中快速查看 SLO 的当前状态。要对 SLI 进行排序，以便将所有运行不正常的 SLI 位于列表顶部，请选择 SLI 状态列，直到运行不正常的 SLI 均位于顶部。

   SLO 表格默认包含以下各列。您可以通过选择列表上方的齿轮图标调整显示列。有关目标、SLI、达标率和间隔的更多信息，请参阅 SLO 概念。

   - SLO 的名称。
   - 目标列显示各个间隔内必须成功达到 SLI 阈值才能达到 SLO 目标的周期百分比。此列还会显示 SLO 的间隔长度。
   - SLI 状态显示应用程序的当前运行状态是否正常。如果 SLO 在当前选定时间范围内的任何时段运行不正常，SLI 状态则显示为运行不正常。
   - 最终达标率是指截至选定时间范围结束达到的达标水平。按此列排序，查看最有可能无法达到的 SLO。
   - 达标率增量是选定时间范围开始和结束时达标水平的差值。增量为负意味着该指标呈下降趋势。按此列排序，查看 SLO 的最新变化趋势。
   - 最终误差预算（%）是该时段内可能有运行不正常的周期但仍能成功实现 SLO 的总时间百分比。如果您将其设为 5%，并且 SLI 在该间隔剩余周期的 5% 或更短的时间内处于运行不正常状态，仍能成功实现 SLO。
   - 误差预算增量是选定时间范围的开始和结束的误差预算差值。增量为负意味着该指标呈衰退趋势。
   - 结束误差预算（时间）是间隔内可能运行不正常但仍能成功实现 SLO 的实际时间。例如，假设为 14 分钟，则如果在剩余间隔内 SLI 运行不正常的时间少于 14 分钟，则仍能成功实现 SLO。
   - 服务、操作和类型列显示设置此 SLO 的服务和操作的信息。
3. 要查看 SLO 的达标率和错误预算图形，请选择 SLO 名称旁边的单选按钮。

   页面顶部的图形显示 SLO 达标率和错误预算状态。此外还显示与此 SLO 关联的 SLI 指标的图形。

4. 要对未达到其目标的 SLO 进行进一步分类，请选择该 SLO 的关联服务名称或操作名称。您将进入详细信息页面，您可以在此进行进一步分类。有关更多信息，请参阅 通过服务详细信息页面查看详细的服务活动和运行状况。
5. 要更改页面所示图表和表格的时间范围，请在屏幕顶部附近选择新的时间范围。

编辑现有 SLO

按照以下步骤编辑现有 SLO。编辑 SLO 时，只能更改阈值、间隔、达标率目标和标签。要更改服务、操作或指标等其他方面，请创建新的 SLO，而非编辑现有 SLO。

更改部分 SLO 核心配置（例如周期或阈值）后，之前有关达标率和运行状况的所有数据点和评估均会失效。可通过这一操作有效删除并重新创建 SLO。

> **Note**
>
> 如果您编辑 SLO，则该 SLO 的关联警报不会自动更新。您可能需要更新警报以使其与 SLO 保持同步。

编辑现有 SLO

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务级别目标（SLO）。
3. 选择要编辑的 SLO 旁边的单选按钮，然后选择操作、编辑 SLO。
4. 进行更改，然后选择保存更改。

删除 SLO

按照以下步骤删除现有 SLO。

> **Note**
>
> 如果您删除 SLO，则该 SLO 的关联警报不会自动删除。您需要自行将其删除。有关更多信息，请参阅 管理警报。

删除 SLO

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择服务级别目标（SLO）。
3. 选择要编辑的 SLO 旁边的单选按钮，然后选择操作、删除 SLO。
4. 选择确认。

综合监控（Canary）

您可以使用 Amazon CloudWatch Synthetics 创建金丝雀。金丝雀是按计划运行的可配置脚本，用于监控端点和 API。金丝雀遵循相同的路线并执行与客户相同的操作，这使您能够持续验证您的客户体验，即使您的应用程序中没有任何客户流量也可以。使用金丝雀，您可以早于客户先行发现问题。

金丝雀是用 Node.js 或 Python 编写的脚本。它们以 Node.js 或 Python 为框架在您的账户中创建 Lambda 函数。金丝雀通过 HTTP 和 HTTPS 两种协议工作。金丝雀使用包含 CloudWatch Synthetics 库的 Lambda 层。该库包含 CloudWatch Synthetics for NodeJS 金丝雀的 NodeJS 版本和CloudWatch Synthetics for Python 金丝雀的 Python 版本。这些层属于 CloudWatch Synthetics 服务账户。库永远不会传输或存储客户信息。所有客户数据都仅存储在客户账户中。

金丝雀通过 Puppeteer 或 Selenium Webdriver 提供对无头 Google Chrome 浏览器的编程访问。有关 Puppeteer 的更多信息，请参阅 Puppeteer。有关 Selenium 的更多信息，请参阅 www.selenium.dev/。

金丝雀检查终端节点的可用性和延迟，并可以存储加载时间数据和 UI 屏幕截图。它们可以监控您的 REST API、URL 和网站内容，并且可以检查来自网络钓鱼、代码注入和跨站脚本的未经授权更改。

CloudWatch Synthetics 与 Application Signals 集成，可以发现和监控您的应用程序服务、客户端、Synthetics Canary 和服务依赖项。使用 Application Signals 查看您的服务列表或可视地图，根据您的服务级别目标（SLO）查看运行状况指标，并深入查看关联 X-Ray 跟踪以便更详细地进行问题排查。要在 Application Signals 中查看您的 Canary，打开 X-Ray 活动跟踪。您的 Canary 显示在连接到您的服务的服务地图，以及所调用服务的服务详细信息页面上。

有关金丝雀的视频演示，请参阅以下内容:

- Amazon CloudWatch Synthetics 简介
- Amazon CloudWatch Synthetics 演示
- 使用 Amazon CloudWatch Synthetics 创建金丝雀
- 使用 Amazon CloudWatch Synthetics 进行可视化监控

您可以运行一次金丝雀，也可以定期运行。金丝雀的运行频率可达每分钟一次。您可以使用 cron 和 rate 表达式来计划金丝雀。

有关在创建和运行金丝雀之前需要考虑的安全问题相关信息，请参阅 Synthetics 金丝雀的安全注意事项。

预设情况下，金丝雀会在 `CloudWatchSynthetics` 命名空间中创建几个 CloudWatch 指标。这些指标使用 `CanaryName` 作为维度。使用函数库中 `executeStep()` 或 `executeHttpStep()` 函数的金丝雀还具有 `StepName` 维度。有关金丝雀函数库的更多信息，请参阅可用于金丝雀脚本的库函数。

CloudWatch Synthetics 与 X-Ray 跟踪地图集成良好，其将 CloudWatch 与 AWS X-Ray 结合使用，以提供服务的端到端视图，帮助您更有效地查明性能瓶颈并确定受影响的用户。使用 CloudWatch Synthetics 创建的 Canary 将显示在跟踪地图中。有关更多信息，请参阅 X-Ray 跟踪地图。

CloudWatch Synthetics 目前在所有商业 AWS 区域和 GovCloud 区域可用。

> **Note**
>
> 在亚太地区（大阪）不支持 AWS PrivateLink。在亚太地区（雅加达），AWS PrivateLink 和 X-Ray 不受支持。

主题

- CloudWatch 金丝雀的必需角色和权限
- 创建金丝雀
- 组
- 在本地测试 Canary
- 排查失败金丝雀的问题
- 金丝雀脚本示例代码
- 金丝雀和 X-Ray 跟踪
- 在 VPC 上运行金丝雀
- 加密金丝雀构件
- 查看金丝雀统计数据和详细信息
- 金丝雀发布的 CloudWatch 指标
- 编辑或删除金丝雀脚本

- 启动、停止、删除或更新多个金丝雀脚本的运行时
- 使用 Amazon EventBridge 监控金丝雀事件

CloudWatch 金丝雀的必需角色和权限

创建和管理金丝雀的用户以及金丝雀本身都必须拥有某些权限。

管理 CloudWatch 金丝雀的用户的必需角色和权限

若要查看 Canary 详细信息和 Canary 运行的结果，您必须以附加了 `CloudWatchSyntheticsFullAccess` 或 `CloudWatchSyntheticsReadOnlyAccess` 策略的用户身份登录。要在控制台中读取所有 Synthetics 数据，您还需要 `AmazonS3ReadOnlyAccess` 和 `CloudWatchReadOnlyAccess` 策略。要查看金丝雀使用的源代码，您还需要 `AWSLambda_ReadOnlyAccess` 策略。

要创建 Canary ，您必须以具有 `CloudWatchSyntheticsFullAccess` 策略或类似权限集的用户身份登录。要为金丝雀创建 IAM 角色，您还需要以下内联策略语句：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

> **Important**
>
> 通过向用户授予 `iam:CreateRole`、`iam:CreatePolicy` 和 `iam:AttachRolePolicy` 权限，用户将获得对 AWS 账户的完全管理访问权限。例如，具有这些权限的用户可以创建一个

对所有资源具有完全权限的策略，并将该策略附加到任何角色。请谨慎地为相关人员授予这些权限。

有关附加策略和向用户授予权限的信息，请参阅更改 IAM 用户的权限和为用户或角色嵌入内联策略。

金丝雀的必需角色和权限

每个金丝雀必须与附加某些权限的 IAM 角色相关联。当您使用 CloudWatch 控制台创建金丝雀时，您可以选择 CloudWatch Synthetics 为金丝雀创建 IAM 角色。如果您这样做，该角色将拥有所需的权限。

如果您要自己创建 IAM 角色，或者创建使用 AWS CLI 或 API 创建金丝雀时可以使用的 IAM 角色，角色必须包含本部分中列出的权限。

金丝雀的所有 IAM 角色必须包含以下信任策略语句。

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

此外，金丝雀的 IAM 角色需要以下语句之一。

不使用 AWS KMS 或者需要 Amazon VPC 访问的基本金丝雀

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"

            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
               "StringEquals": {
                "cloudwatch:namespace": "CloudWatchSynthetics"
```

```
                }
            }
        }
    ]
}
```

使用 AWS KMS 加密金丝雀构件但不需要 Amazon VPC 访问的金丝雀

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

不使用 AWS KMS 但需要 Amazon VPC 访问的金丝雀

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
```

```
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
```

```
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
       }
    ]
}
```

使用 AWS KMS 加密金丝雀构件并且也需要 Amazon VPC 访问的金丝雀

如果您更新非 VPC 金丝雀以开始使用 VPC，则为了包含以下策略中列出的网络接口权限，需要更新金丝雀的角色。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
```

```
        },
        {
             "Effect": "Allow",
             "Action": [
                 "kms:Decrypt",
                 "kms:GenerateDataKey"
             ],
             "Resource":
 "arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
             "Condition": {
                 "StringEquals": {
                     "kms:ViaService": [
                         "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
                     ]
                 }
             }
         }
    ]
}
```

用于 CloudWatch Synthetics 的 AWS 托管式策略

要向用户、组和角色添加权限，与自己编写策略相比，使用 AWS 托管策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管式策略的更多信息，请参阅 IAM 用户指南中的 AWS 托管式策略 AWS 托管式策略。

AWS 服务负责维护和更新 AWS 管理型策略。您无法更改 AWS 管理型策略中的权限。服务会偶尔更改 AWS 托管式策略中的权限。此类更新会影响附加策略的所有身份（用户、组和角色）。

CloudWatch Synthetics 对 AWS 托管式策略的更新

查看有关 CloudWatch Synthetics 的 AWS 托管式策略更新的详细信息（从该服务开始跟踪这些更改开始）。有关此页面更改的自动提示，请订阅 CloudWatch 文档历史记录页面上的 RSS 源。

更改	描述	日期
删除了 CloudWatchSyntheticsFullAccess 中的冗余操作	CloudWatch Synthetics 删除了 CloudWatchSyntheticsFullAccess 策略中的 `s3:PutBucketEncryption` 和	2021 年 3 月 12 日

更改	描述	日期
	`lambda:GetLayerVersionByArn` 操作，因为这些操作相较于策略中的其他权限是多余的。删除的操作不提供任何权限，策略授予的权限也未发生任何净更改。	
CloudWatch Synthetics 开始跟踪变更	CloudWatch Synthetics 开始跟踪其 AWS 托管式策略的更改。	2021 年 3 月 10 日

CloudWatchSyntheticsFullAccess

以下是 `CloudWatchSyntheticsFullAccess` 策略的内容：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "synthetics:*"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:CreateBucket",
            "s3:PutEncryptionConfiguration"
         ],
         "Resource":[
            "arn:aws:s3:::cw-syn-results-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:ListRoles",
```

```
            "s3:ListAllMyBuckets",
            "s3:GetBucketLocation",
            "xray:GetTraceSummaries",
            "xray:BatchGetTraces",
            "apigateway:GET"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject",
            "s3:ListBucket"
         ],
         "Resource":"arn:aws:s3:::cw-syn-*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObjectVersion"
         ],
         "Resource":"arn:aws:s3:::aws-synthetics-library-*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:PassRole"
         ],
         "Resource":[
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
         ],
         "Condition":{
            "StringEquals":{
               "iam:PassedToService":[
                  "lambda.amazonaws.com",
                  "synthetics.amazonaws.com"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "iam:GetRole"
```

```
         ],
         "Resource":[
            "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:GetMetricData",
            "cloudwatch:GetMetricStatistics"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:PutMetricAlarm",
            "cloudwatch:DeleteAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "cloudwatch:DescribeAlarms"
         ],
         "Resource":[
            "arn:aws:cloudwatch:*:*:alarm:*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "lambda:CreateFunction",
            "lambda:AddPermission",
            "lambda:PublishVersion",
            "lambda:UpdateFunctionConfiguration",
            "lambda:GetFunctionConfiguration"
         ],
         "Resource":[
            "arn:aws:lambda:*:*:function:cwsyn-*"
         ]
```

```
      },
      {
         "Effect":"Allow",
         "Action":[
            "lambda:GetLayerVersion",
            "lambda:PublishLayerVersion"
         ],
         "Resource":[
            "arn:aws:lambda:*:*:layer:cwsyn-*",
            "arn:aws:lambda:*:*:layer:Synthetics:*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sns:ListTopics"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "sns:CreateTopic",
            "sns:Subscribe",
            "sns:ListSubscriptionsByTopic"
         ],
         "Resource":[
            "arn:*:sns:*:*:Synthetics-*"
         ]
      }
   ]
```

```
}
```

CloudWatchSyntheticsReadOnlyAccess

以下是 CloudWatchSyntheticsReadOnlyAccess 策略的内容：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

限制用户查看特定的金丝雀

您可以限制用户查看金丝雀相关信息，使他们只能看到您指定金丝雀的信息。为此，请将 IAM 策略与类似以下内容的 Condition 声明配合使用，并将此策略附加到用户或 IAM 角色。

以下示例限制了用户只能查看 name-of-allowed-canary-1 和 name-of-allowed-canary-2 相关信息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "synthetics:Names": [
                        "name-of-allowed-canary-1",
                        "name-of-allowed-canary-2"
                    ]
                }
```

```
            }
        }
    ]
}
```

CloudWatch Synthetics 支持在 `synthetics:Names` 数组中最多列出五个项目。

您也可以创建策略，在允许的金丝雀名称中使用 * 作为通配符，如以下示例所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringLike": {
                    "synthetics:Names": [
                        "my-team-canary-*"
                    ]
                }
            }
        }
    ]
}
```

使用所附的这些策略之一登录的任何用户都不能使用 CloudWatch 控制台查看任何金丝雀信息。他们只能使用 DescribeCanaries API 或 describe-canaries AWS CLI 命令，查看策略授权的金丝雀信息。

创建金丝雀

> **Important**
>
> 确保使用 Synthetics 金丝雀来仅监控您拥有所有权或权限的终端节点和 API。根据金丝雀运行频率设置，这些端点可能会面临流量增加的情况。

当您使用 CloudWatch 控制台创建金丝雀时，您可以使用 CloudWatch 提供的蓝图创建您的金丝雀，也可以编写您自己的脚本。有关更多信息，请参阅 使用金丝雀蓝图。

如果使用的是您自己的金丝雀脚本，您还可以使用 AWS CloudFormation 创建金丝雀。有关更多信息，请参阅 AWS CloudFormation 用户指南中的 AWS::Synthetics::Canary。

如果您正在编写自己的脚本，则可以使用 CloudWatch Synthetics 已内置在库中的多个函数。有关更多信息，请参阅 Synthetics 运行时版本。

> **Note**
>
> 当您创建金丝雀时，所创建的其中一个层是前面加上 `Synthetics` 的 Synthetics 层。该层归 Synthetics 服务账户所有，且包含运行时系统代码。

创建金丝雀

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。
3. 选择 Create Canary (创建金丝雀)。
4. 选择以下操作之一：
   - 要将某个蓝图脚本作为金丝雀的基础，请选择 Use a blueprint (使用蓝图)，然后选择要创建的金丝雀的类型。有关每种类型的蓝图执行的操作的更多信息，请参阅使用金丝雀蓝图。
   - 要上传您自己的 node.js 脚本以创建自定义金丝雀，请选择 Upload a script (上传脚本)。

     然后，您可以将脚本拖入 Script (脚本) 区域，或者选择 Browse files (浏览文件)，在文件系统中浏览找到该脚本。
   - 要从 S3 存储桶导入脚本，请选择 Import from S3 (从 S3 导入)。然后，在 Source location (源位置) 下，输入金丝雀的完整路径或选择 Browse S3 (浏览 S3)。

     您必须对您使用的 S3 存储桶具有 `s3:GetObject` 和 `s3:GetObjectVersion` 权限。存储桶必须位于您在其中创建金丝雀的同一 AWS 区域内。
5. 在 Name (名称) 下，输入金丝雀的名称。此名称将用于许多页面，因此，我们建议您为该金丝雀提供一个描述性名称，以将它与其他金丝雀区分开来。
6. 在 Application or endpoint URL (应用程序或终端节点 URL) 下，输入您希望金丝雀测试的 URL。此 URL 必须包含协议（例如 https:// ）。

   如果您希望金丝雀测试 VPC 上的终端节点，还必须在此过程随后的步骤中输入有关 VPC 的信息。

7. 如果您将自己的脚本用于金丝雀，请在 Lambda handler (Lambda 处理程序) 下，输入您希望金丝雀开始的入口点。如果您使用 `syn-nodejs-puppeteer-3.4` 或 `syn-python-selenium-1.1` 之前版本的运行时，则您输入的字符串必须以 `.handler` 结尾。如果您使用 `syn-nodejs-puppeteer-3.4` 或 `syn-python-selenium-1.1` 或之后版本的运行时，则此限制不适用。
8. 如果在脚本中使用环境变量，请选择 Environment variables（环境变量），然后为脚本中定义的每个环境变量指定值。有关更多信息，请参阅 环境变量。
9. 在 Schedule（计划）下，选择是运行此金丝雀一次、使用 rate 表达式连续运行，还是使用 cron 表达式对其进行计划。

   - 当您使用 CloudWatch 控制台创建连续运行的金丝雀时，您可以在每分钟一次到每小时一次之间的范围内选择一个运行速率。

   - 有关为金丝雀调度编写 cron 表达式的更多信息，请参阅 使用 cron 安排金丝雀运行。
10. （可选）要为金丝雀设置超时值，请选择 Additional configuration（其它配置），然后指定该超时值。此值不要短于 15 秒钟，以预留 Lambda 冷启动以及启动金丝雀工具的时间。
11. 在 Data retention (数据保留) 下，指定保留有关失败和成功的金丝雀运行的信息的时间。范围为 1 - 455 天。

    此设置仅影响 CloudWatch Synthetics 存储和显示在控制台中的数据。它不会影响 Amazon S3 存储桶中存储的数据，也不会影响 Canary 发布的日志或指标。
12. 在 Data Storage (数据存储) 下，选择要用于存储来自金丝雀运行的数据的 S3 存储桶。存储桶名称不能包含句点 (.)。如果将此选项保留空白，将使用或创建默认 S3 存储桶。

    如果您使用的是 `syn-nodejs-puppeteer-3.0` 或更高版本的运行时，在文本框中输入存储桶的 URL 时，您可以指定当前区域或其他区域中的存储桶。如果您使用的是较早版本的运行时，则存储桶必须位于当前区域中。
13. （可选）默认情况下，金丝雀将自己的构件存储在 Amazon S3 上，并使用 AWS 托管式 AWS KMS 密钥将构件静态加密。您可以通过选择 Data Storage（数据存储）部分的 Additional configuration（其他配置）来使用不同的加密选项。然后，您可以选择用于加密的密钥类型。有关更多信息，请参阅 加密金丝雀构件。
14. 在 Access permissions（访问权限）下，选择是创建 IAM 角色来运行金丝雀，还是使用现有角色。

    如果您使用 CloudWatch Synthetics 创建角色，则其会自动包含所有必要的权限。如果您要自己创建角色，请参阅 金丝雀的必需角色和权限 获取有关必要权限的信息。

如果您在创建金丝雀时使用 CloudWatch 控制台为金丝雀创建角色，则无法将该角色重新用于其他金丝雀，因为这些角色仅特定于一个金丝雀。如果您已手动创建了适用于多个金丝雀的角色，则可以使用现有角色。

若要使用现有角色，您必须具有 `iam:PassRole` 权限，以将角色传递到 Synthetics 和 Lambda。您还必须拥有 `iam:GetRole` 权限。

15. （可选）在 Alarms（告警）下，选择是否要为此金丝雀创建默认 CloudWatch 告警。如果选择创建告警，则会按照以下命名约定创建告警：`Synthetics-Alarm-`*`canaryName`*`-`*`index`*

    `index` 是一个数字，代表为此金丝雀创建的各个不同的告警。第一个告警的索引为 1，第二个告警的索引为 2，依此类推。

16. （可选）要让此金丝雀测试 VPC 上的终端节点，请选择 VPC settings (VPC 设置)，并执行以下操作：

    a. 选择托管终端节点的 VPC。

    b. 选择 VPC 上的一个或多个子网。您必须选择私有子网，因为在执行过程中无法为 Lambda 实例分配 IP 地址时，无法将 Lambda 实例配置为在公有子网中运行。有关更多信息，请参阅[配置 Lambda 函数以访问 VPC 中的资源]。

    c. 在 VPC 上选择一个或多个安全组。

    如果端点位于 VPC 上，则必须启用您的金丝雀才能向 CloudWatch 和 Amazon S3 发送信息。有关更多信息，请参阅 [在 VPC 上运行金丝雀]。

17. （可选）在 Tags (标签) 下，添加一个或多个键/值对作为此金丝雀的标签。标签可帮助您识别和组织 AWS 资源并跟踪 AWS 成本。有关更多信息，请参阅 [标记 Amazon CloudWatch 资源]。

18. （可选）在 Active tracing（活动跟踪）下，选择是否启用此金丝雀的活动 X-Ray 跟踪。仅当金丝雀使用 syn-nodejs-2.0 或更高的运行时版本时，此选项才可用。有关更多信息，请参阅 [金丝雀和 X-Ray 跟踪]。

为金丝雀创建的资源

创建金丝雀时，将创建以下资源：

- 一个名为 `CloudWatchSyntheticsRole-`*`canary-name`*`-`*`uuid`* 的 IAM 角色（如果您使用 CloudWatch 控制台创建金丝雀并指定为金丝雀创建新角色）
- 一个名为 `CloudWatchSyntheticsPolicy-`*`canary-name`*`-`*`uuid`* 的 IAM 策略。

- 一个名为 cw-syn-results-*accountID*-*region* 的 S3 存储桶。
- 名为 Synthetics-Alarm-*MyCanaryName* 的警报（如果您为要为金丝雀创建警报）。
- Lambda 函数和层（如果您使用蓝图创建金丝雀）。这些资源具有前缀 cwsyn-*MyCanaryName*。
- 名为 /aws/lambda/cwsyn-*MyCanaryName*-*randomId* 的 CloudWatch Logs 日志组。

使用金丝雀蓝图

本节提供有关每种金丝雀蓝图的详细信息以及每种蓝图最适合的任务。为以下金丝雀类型提供了蓝图：

- 检测信号监控器
- API 金丝雀
- 无效链接检查器
- 可视化监控
- 金丝雀记录器
- GUI 工作流程

当您使用蓝图创建金丝雀时，在填写 CloudWatch 控制台中的字段时，页面的 Script editor（脚本编辑器）区域会将您正在创建的金丝雀显示为 Node.js 脚本。您还可以在此区域编辑金丝雀以进一步定制。

检测信号监控

检测信号脚本加载指定的 URL，并存储页面的屏幕截图和 HTTP 归档文件（HAR 文件）。它们还存储已访问 URL 的日志。

您可以使用 HAR 文件查看有关网页性能的详细数据。您可以分析 Web 请求列表并捕获性能问题，例如某个项的加载时间。

如果您的金丝雀使用 syn-nodejs-puppeteer-3.1 或更高版本的运行时，您可以使用检测信号监控蓝图，以监控多个 URL 并查看状态、持续时间、关联的屏幕截图和金丝雀运行报告的步骤摘要中各个 URL 的故障原因。

API 金丝雀

API 金丝雀可以测试 REST API 的基本读写函数。REST 指表述性状态转移，是开发人员在创建 API 时需要遵循的一组规则。其中一条规则规定，指向特定 URL 的链接应返回一段数据。

金丝雀可与任何 API 结合使用并测试所有类型的功能。每个金丝雀都可以做出多个 API 调用。

在使用 syn-nodejs-2.2 或更高版本运行时的金丝雀中，API 金丝雀蓝图支持将 API 作为 HTTP 步骤进行监控的多步骤金丝雀。您可以在单个金丝雀中测试多个 API。每个步骤都是一个单独的请求，可以访问不同的 URL、使用不同的标头以及对是否捕获标题和响应正文使用不同的规则。通过不捕获标头和响应正文，可以防止敏感数据被记录下来。

API 金丝雀中的每个请求均包含以下信息：

- 终端节点，即您请求的 URL。
- 方法，它是发送到服务器的请求类型。REST API 支持 GET（读取）、POST（写入）、PUT（更新）、PATCH（更新）和 DELETE（删除）操作。
- 标头，用于向客户端和服务器提供信息。它们用于身份验证和提供有关正文内容的信息。有关有效标头的列表，请参阅 HTTP 标头。
- 数据（或正文），其中包含要发送到服务器的信息。这仅用于 POST、PUT、PATCH 或 DELETE 请求。

API 金丝雀蓝图支持 GET 和 POST 方法。使用此蓝图时，您必须指定标头。例如，您可以指定 **Authorization** 作为 Key (键)，并指定必要的授权数据作为该键的 Value (值)。

如果您在测试 POST 请求，还可以在 Data (数据) 字段中指定要发布的内容。

与 API Gateway 的集成

API 蓝图与 Amazon API Gateway 集成。因此，您能够从与该金丝雀所在的同一个 AWS 账户和区域选择一个 API Gateway API 和阶段，或上载 API Gateway 中的 Swagger 模板以进行跨账户和跨区域 API 监控。然后，您可以在控制台中选择其余详细信息来创建金丝雀，而不是从 Scratch 输入这些详细信息。有关 API Gateway 的更多信息，请参阅什么是 Amazon API Gateway？。

使用私有 API

您可以在 Amazon API Gateway 中创建使用私有 API 的金丝雀。有关更多信息，请参阅 在 Amazon API Gateway 中创建私有 API?

失效链接检查器

无效链接检查器通过使用 document.getElementsByTagName('a') 收集您正在测试的 URL 内的所有链接。它仅测试您指定数量的链接，而且 URL 本身计为第一个链接。例如，如果要检查包含五个链接的页面上的所有链接，则必须指定金丝雀跟踪六个链接。

使用 syn-nodejs-2.0-beta 或更高版本运行时创建的无效链接检查器金丝雀支持以下额外功能：

- 提供包含所检查链接、状态代码、故障原因（如有）以及源页面和目标页面屏幕截图的报告。
- 查看金丝雀结果时，您可以筛选以仅查看无效链接，然后根据故障原因修复链接。
- 此版本会捕获每个链接的带注释源页面屏幕截图，并突出显示链接所在的锚点。不会对隐藏组件添加注释。
- 您可以将此版本配置为捕获源页面和目标页面的屏幕截图、仅捕获源页面或仅捕获目标页面的屏幕截图。
- 此版本修复了早期版本中的一个问题，在早期版本中，即使从第一页抓取了更多链接，金丝雀脚本也会在第一个无效链接后停止。

如果您希望使用 `syn-1.0` 来更新现有金丝雀以使用新的运行时，则必须删除并重新创建金丝雀。这些功能并不会随着将现有金丝雀更新到新运行时而变得可用。

无效链接检查器金丝雀可检测以下类型的链接错误：

- 404 未找到页面
- 主机名无效
- 错误的 URL。例如，URL 缺少括号、包含多余的斜线，或者协议错误。
- 无效的 HTTP 响应代码。
- 主机服务器返回没有内容和没有响应代码的空响应。
- HTTP 请求在金丝雀运行期间持续超时。
- 主机持续丢弃连接，因为它配置错误或太忙。

可视监控蓝图

可视化监控蓝图包含用以比较在金丝雀运行期间捕获的屏幕截图与在基准金丝雀运行期间捕获的屏幕截图的代码。如果两个屏幕截图之间的差异超过阈值百分比，则金丝雀将失败。运行 syn-puppeteer-node-3.2 和更高版本的金丝雀支持可视化监控。运行 Python 和 Selenium 的金丝雀中目前不支持可视化监控。

可视化监控蓝图在默认蓝图金丝雀脚本中包含以下代码行，这些代码可实现可视化监控。

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

将此行添加到脚本后，金丝雀首次成功运行时，它会使用在该运行期间捕获的屏幕截图作为比较基准。在金丝雀的该首次运行之后，您可以使用 CloudWatch 控制台编辑金丝雀以执行以下任一操作：

- 将金丝雀的下一次运行设置为新基准。
- 在当前基准屏幕截图上绘制边界，以指定在可视化比较过程中要忽略的屏幕截图区域。
- 删除屏幕截图，使其不用于可视化监控。

有关使用 CloudWatch 控制台编辑金丝雀的更多信息，请参阅 编辑或删除金丝雀脚本。

您还可以通过使用 `nextrun` 或 `lastrun` 参数或在 UpdateCanary API 中指定金丝雀运行 ID 来更改用作基准的金丝雀运行。

使用可视化监控蓝图时，您可以输入要在其中捕获屏幕截图的 URL，并将差值阈值指定为百分比。在基准运行之后，只要在金丝雀的未来运行中检测到大于该阈值的可视化差异，便会触发金丝雀故障。在基准运行之后，您还可以编辑金丝雀，以在可视化监控过程中要忽略的基准屏幕截图上“绘制”边界。

可视化监控功能由 ImageMagick 开源软件工具包提供支持。有关更多信息，请参阅 ImageMagick。

金丝雀记录器

使用金丝雀记录器蓝图，您可以使用 CloudWatch Synthetics Recorder 记录您的点击并在网站上键入操作，并自动生成可用于创建按相同步骤操作的金丝雀的 Node.js 脚本。CloudWatch Synthetics Recorder 是 Amazon 推出的 Google Chrome 扩展程序。

开发团队：CloudWatch Synthetics Recorder 基于无头记录器。

有关更多信息，请参阅 使用面向 Google Chrome 的 CloudWatch Synthetics Recorder。

GUI 工作流生成器

GUI 工作流生成器蓝图验证是否可以在您的网页上执行操作。例如，如果您有一个带有登录表单的网页，则金丝雀可以填充用户和密码字段并提交表单，以验证网页是否正常工作。

当您使用蓝图创建此类型的金丝雀时，可以指定希望金丝雀在网页上执行的操作。您可以使用的操作如下：

- 单击 – 选择您要指定的元素，然后模拟用户单击或选择该元素的行为。

  若要在 Node.js 脚本中指定该元素，请使用 `[id=]` 或 `a[class=]`。

  若要在 Python 脚本中指定该元素，请使用 `xpath //*[@id=]` 或 `//*[@class=]`。
- 验证选择器 – 验证网页上存在指定的元素。此测试对于验证之前的操作是否导致使用正确的元素填充页面非常有用。

若要在 Node.js 脚本中指定待验证的元素，请使用 `[id=]` 或 `a[class=]`。

若要在 Python 脚本中指定待验证的元素，请使用 `xpath //*[@id=]` 或 `//*[class=]`。

- 验证文本 – 验证指定的字符串是否包含在目标元素中。此测试对于验证之前的操作是否导致显示正确的文本非常有用。

  若要在 Node.js 脚本中指定元素，请使用诸如 `div[@id=]//h1` 之类的格式，因为此操作使用 Puppeteer 中的 `waitForXPath` 函数。

  若要在 Python 脚本中指定元素，请使用 xpath 格式（例如 `//*[@id=]` 或 //*[@class=]），因为此操作使用 Selenium 中的 `implicitly_wait` 函数。

- 输入文本 – 在目标元素中写入指定的文本。

  若要在 Node.js 脚本中指定待验证的元素，请使用 `[id=]` 或 `a[class=]`。

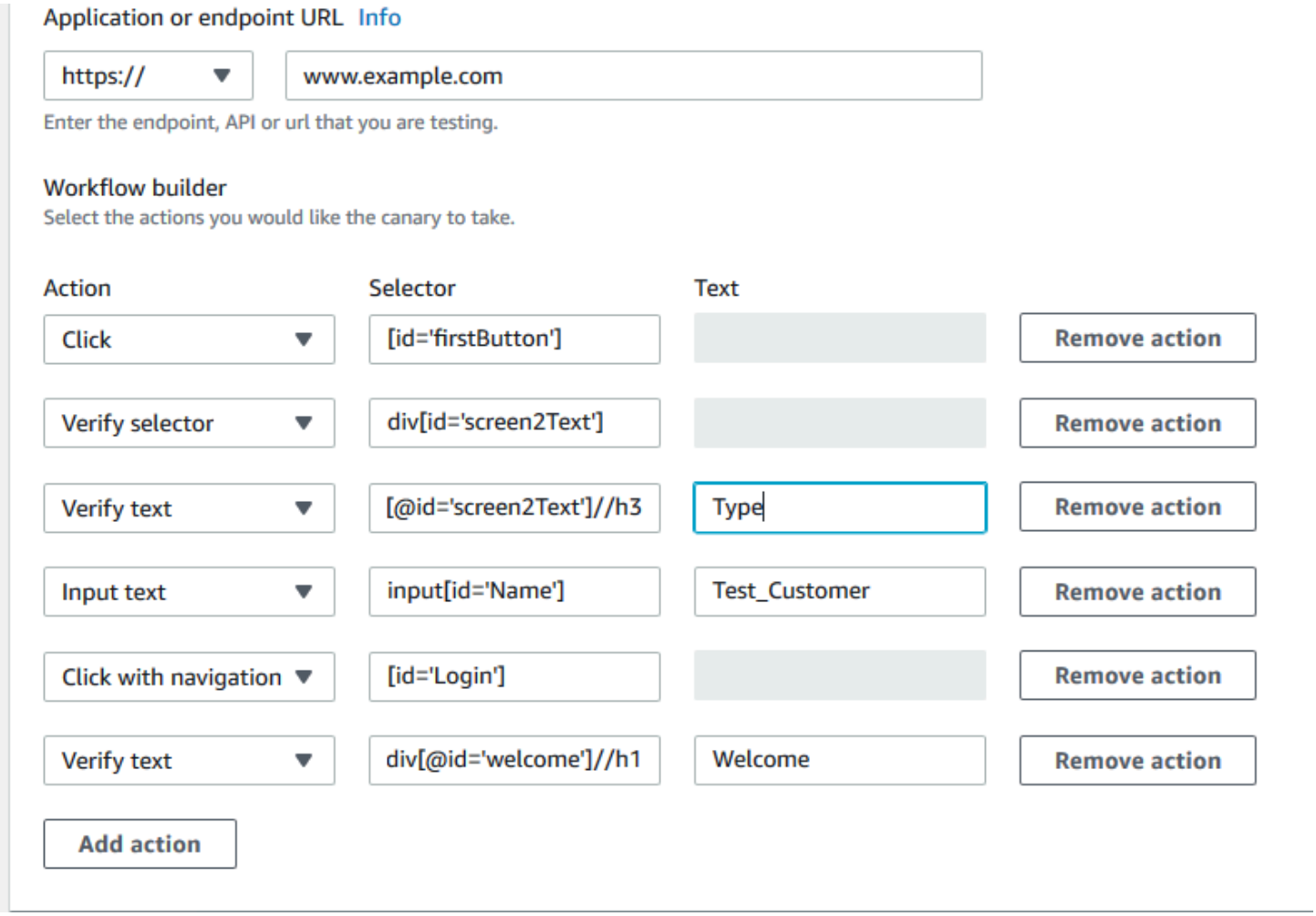
  若要在 Python 脚本中指定待验证的元素，请使用 `xpath //*[@id=]` 或 `//*[@class=]`。

- 单击并导航 – 在选择指定的元素之后，等待整个页面加载。当您需要重新加载页面时，这非常有用。

  若要在 Node.js 脚本中指定该元素，请使用 `[id=]` 或 `a[class=]`。

  若要在 Python 脚本中指定该元素，请使用 `xpath //*[@id=]` 或 `//*[@class=]`。

例如，以下蓝图使用 Node.js。其会单击指定 URL 上的 firstButton，验证预期的选择器是否出现了预期文本，将名称 `Test_Customer` 输入到 Name（名称）字段中，单击 Login（登录）按钮，然后通过检查下一页上的 Welcome（欢迎）文本来验证登录是否成功。



使用以下运行时的 GUI 工作流金丝雀还提供每次金丝雀运行所执行的步骤的摘要。您可以使用与每个步骤关联的屏幕截图和错误消息来查找故障的根本原因。

- `syn-nodejs-2.0` 或更高版本
- `syn-python-selenium-1.0` 或更高版本

使用面向 Google Chrome 的 CloudWatch Synthetics Recorder

Amazon 推出 CloudWatch Synthetics Recorder，以帮助您更轻松地创建金丝雀。此记录器是一个 Google Chrome 扩展程序。

此记录器记录您在网站上的单击和键入操作，并自动生成可用于创建按相同步骤操作的金丝雀的 Node.js 脚本。

开始记录后，CloudWatch Synthetics Recorder 会在浏览器中检测您的操作并将其转换为脚本。您可以根据需要暂停和恢复记录。当您停止记录时，记录器会将您的操作生成 Node.js 脚本，您可以使用

Copy to Clipboard（复制到剪贴板）按钮轻松复制这些脚本。之后，您可以使用此脚本在 CloudWatch Synthetics 中创建金丝雀。

开发团队：CloudWatch Synthetics Recorder 基于无头记录器。

安装面向 Google Chrome 的 CloudWatch Synthetics Recorder 扩展程序

若要使用 CloudWatch Synthetics Recorder，您可以先创建一个金丝雀，然后选择 Canary Recorder（金丝雀记录器）蓝图。如果您在尚未下载记录器时执行此操作，则 CloudWatch Synthetics 控制台会提供下载该记录器的链接。

您也可以按照以下步骤直接下载并安装该记录器。

安装 CloudWatch Synthetics Recorder

1. 使用 Google Chrome，访问此网站：https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkkdfplenecpegfno
2. 选择 Add to Chrome（添加到 Chrome），然后选择 Add extension（添加扩展程序）。

使用面向 Google Chrome 的 CloudWatch Synthetics Recorder

若要使用 CloudWatch Synthetics Recorder 帮助您创建金丝雀，您可以在 CloudWatch 控制台中选择 Create canary（创建金丝雀），然后依次选择 Use a blueprint（使用蓝图）、Canary Recorder（金丝雀记录器）。有关更多信息，请参阅 创建金丝雀。

您也可以使用记录器记录步骤，而不必立即使用它们创建金丝雀。

使用 CloudWatch Synthetics Recorder 记录您在网站上的行为

1. 导航到要监控的页面。
2. 选择 Chrome 扩展程序图标，然后选择 CloudWatch Synthetics Recorder。
3. 选择 Start Recording（开始记录）。
4. 执行您希望记录的步骤。若要暂停记录，请选择 Pause（暂停）。
5. 完成记录工作流后，选择 Stop recording（停止记录）。
6. 选择 Copy to clipboard（复制到剪贴板）以将生成的脚本复制到剪贴板。或者，如果要重新开始，请选择 New recording（新建记录）。
7. 若要使用复制的脚本创建金丝雀，您可以将复制的脚本粘贴到记录器蓝图内联编辑器中，或者将其保存到 Amazon S3 存储桶中，然后从该存储桶中导入。

8. 如果您不立即创建金丝雀，则可以将记录的脚本保存到文件中。

CloudWatch Synthetics Recorder 的已知局限性

面向 Google Chrome 的 CloudWatch Synthetics Recorder 目前具有以下局限性。

- 不具有 ID 的 HTML 元素将使用 CSS 选择器。如果网页结构稍后发生变化，这可能会破坏金丝雀。我们计划在未来版本的记录器中针对此问题提供一些配置选项（例如使用 data-id）。
- 记录器不支持双击或复制/粘贴等操作，也不支持 CMD+0 等组合键。
- 若要验证页面上是否存在某元素或文本，用户必须在脚本后生成添加断言。记录器不支持在不对某元素执行任何操作的情况下验证该元素。这与金丝雀工作流生成器中的“验证文本”或“验证元素”选项类似。我们计划在未来版本的记录器中添加一些断言支持。
- 记录器会记录启动记录的选项卡中的所有操作。其不会记录弹出窗口（例如，允许位置跟踪的弹出窗口）或从弹出窗口到不同页面的导航。

Synthetics 运行时版本

创建或更新金丝雀时，您可以为金丝雀选择 Synthetics 运行时版本。Synthetics 运行时是调用脚本处理程序的 Synthetics 代码，以及捆绑依赖关系的 Lambda 层的组合。

CloudWatch Synthetics 目前支持将 Node.js 用于脚本和 Puppeteer 框架的运行时，以及使用 Python 编写脚本和将 Selenium Webdriver 用于框架的运行时。

我们建议您始终为金丝雀使用最新的运行时版本，以便能够使用最新的功能和对 Synthetics 库进行的更新。

当您创建金丝雀时，所创建的其中一个层是前面加上 `Synthetics` 的 Synthetics 层。该层归 Synthetics 服务账户所有，且包含运行时系统代码。

> **Note**
>
> 每当您升级金丝雀以使用新版本的 Synthetics 运行时时，您的金丝雀使用的所有 Synthetics 库函数也会自动升级到 Synthetics 运行时支持的相同 NodeJS 版本。

主题

- CloudWatch Synthetics 运行时支持策略
- 使用 Node.js 和 Puppeteer 的运行时版本

- 使用 Python 和 Selenium Webdriver 的运行时版本

CloudWatch Synthetics 运行时支持策略

Synthetics 运行时版本受维护和安全更新的约束。如果不再支持运行时版本的任何组件，则该 Synthetics 运行时版本将被弃用。

您无法使用已弃用的运行时版本创建金丝雀。使用已弃用运行时的金丝雀将继续运行。您可以停止、启动和删除这些金丝雀。您可以通过将金丝雀更新为使用受支持的运行时版本，来更新使用已弃用运行时版本的现有金丝雀。

如果您有使用计划将在未来 60 天内弃用的运行时的金丝雀，则 CloudWatch Synthetics 会通过电子邮件通知您。我们建议您将金丝雀迁移到受支持的运行时版本，以享受最新版本中包含的新功能、安全性和性能增强的益处。

如何将金丝雀更新到新的运行时版本？

您可以使用 CloudWatch 控制台、AWS CloudFormation、AWS CLI 或 AWS SDK 更新金丝雀的运行时版本。当使用 CloudWatch 控制台更新时，您可以一次最多更新五个金丝雀，方法是在金丝雀列表页面中选中它们，然后选择 Actions（操作）、Update Runtime（更新运行时）。

要验证升级，您可以先使用 CloudWatch 控制台克隆金丝雀，然后更新其运行时版本。这会再创建一个金丝雀，其是原始金丝雀的克隆。在使用新的运行时版本验证了金丝雀后，您便可以更新原始金丝雀的运行时版本并删除克隆金丝雀。

您还可以使用升级脚本更新多个金丝雀。有关更多信息，请参阅 金丝雀运行时升级脚本。

如果升级金丝雀失败，请参阅 排查失败金丝雀的问题。

运行时弃用日期

运行时版本	弃用日期
syn-nodejs-puppeteer-6.1	2024 年 3 月 8 日
syn-nodejs-puppeteer-6.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-5.1	2024 年 3 月 8 日

运行时版本	弃用日期
syn-nodejs-puppeteer-5.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-4.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-3.9	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.8	2024 年 1 月 8 日
syn-python-selenium-2.0	2024 年 3 月 8 日
syn-python-selenium-1.3	2024 年 3 月 8 日
syn-python-selenium-1.2	2024 年 3 月 8 日
syn-python-selenium-1.1	2024 年 3 月 8 日
syn-python-selenium-1.0	2024 年 3 月 8 日
syn-nodejs-puppeteer-3.7	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.6	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.5	2024 年 1 月 8 日
syn-nodejs-puppeteer-3.4	2022 年 11 月 13 日

运行时版本	弃用日期
syn-nodejs-puppeteer-3.3	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.2	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.1	2022 年 11 月 13 日
syn-nodejs-puppeteer-3.0	2022 年 11 月 13 日
syn-nodejs-2.2	2021 年 5 月 28 日
syn-nodejs-2.1	2021 年 5 月 28 日
syn-nodejs-2.0	2021 年 5 月 28 日
syn-nodejs-2.0-beta	2021 年 2 月 8 日
syn-1.0	2021 年 5 月 28 日

金丝雀运行时升级脚本

若要将金丝雀脚本升级到支持的运行时版本，请使用以下脚本。

```
const AWS = require('aws-sdk');

// You need to configure your AWS credentials and Region.
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-credentials-node.html
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
  /**
```

```
    * The number of canaries to upgrade during a single run of this script.
    */
  count: 10,
  /**
   * No canaries are upgraded unless force is specified.
   */
  force: false
};

/**
 * The number of milliseconds to sleep between GetCanary calls when
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
  try {
    const options = getOptions();

    const versions = await getRuntimeVersions();
    const canaries = await getAllCanaries();
    const upgrades = canaries
      .filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
      .map(canary => {
        return {
          Name: canary.Name,
          FromVersion: canary.RuntimeVersion,
          ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
        };
      });

    if (options.force) {
      const promises = [];

      for (const upgrade of upgrades.slice(0, options.count)) {
        const promise = upgradeCanary(upgrade);
        promises.push(promise);
        // Sleep for 100 milliseconds to avoid throttling.
        await usleep(100);
      }

      const succeeded = [];
      const failed = [];
      for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
```

```
        const upgrade = upgrades[i];
        const promise = promises[i];
        try {
          await promise;
          console.log(`The update of ${upgrade.Name} succeeded.`);
          succeeded.push(upgrade.Name);
        } catch (e) {
          console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
          failed.push({
            Name: upgrade.Name,
            Reason: e
          });
        }
      }

      if (succeeded.length) {
        console.group('The following canaries were upgraded successfully.');
        for (const name of succeeded) {
          console.log(name);
        }
        console.groupEnd()
      } else {
        console.log('No canaries were upgraded successfully.');
      }

      if (failed.length) {
        console.group('The following canaries were not upgraded successfully.');
        for (const failure of failed) {
          console.log('\x1b[31m', `${failure.Name}: ${failure.Reason}`, '\x1b[0m');
        }
        console.groupEnd();
      }
    } else {
      console.log('Run with --force [--count <count>] to perform the first <count>
 upgrades shown. The default value of <count> is 10.')
      console.table(upgrades);
    }
  } catch (e) {
    console.error(e);
  }
})();

function getOptions() {
  const force = getFlag('--force', DEFAULT_OPTIONS.force);
```

```
  const count = getOption('--count', DEFAULT_OPTIONS.count);
  return { force, count };

  function getFlag(key, defaultValue) {
    return process.argv.includes(key) || defaultValue;
  }
  function getOption(key, defaultValue) {
    const index = process.argv.indexOf(key);
    if (index < 0) {
      return defaultValue;
    }
    const value = process.argv[index + 1];
    if (typeof value === 'undefined' || value.startsWith('-')) {
      throw `The ${key} option requires a value.`;
    }
    return value;
  }
}

function getAllCanaries() {
  return new Promise((resolve, reject) => {
    const canaries = [];

    synthetics.describeCanaries().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          resolve(canaries);
        } else {
          canaries.push(...data.Canaries);
        }
      }
    });
  });
}

function getRuntimeVersions() {
  return new Promise((resolve, reject) => {
    const jsVersions = [];
    const pythonVersions = [];
    synthetics.describeRuntimeVersions().eachPage((err, data) => {
      if (err) {
        reject(err);
```

```
      } else {
        if (data === null) {
          jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          resolve({
            isLatestVersion(version) {
              const latest = this.getLatestVersion(version);
              return latest === version;
            },
            getLatestVersion(version) {
              if (jsVersions.some(v => v.VersionName === version)) {
                return jsVersions[jsVersions.length - 1].VersionName;
              } else if (pythonVersions.some(v => v.VersionName === version)) {
                return pythonVersions[pythonVersions.length - 1].VersionName;
              } else {
                throw Error(`Unknown version ${version}`);
              }
            }
          });
        } else {
          for (const version of data.RuntimeVersions) {
            if (version.VersionName === 'syn-1.0') {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-python-selenium-')) {
              pythonVersions.push(version);
            } else {
              throw Error(`Unknown version ${version.VersionName}`);
            }
          }
        }
      }
    });
  });
}

async function upgradeCanary(upgrade) {
  console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
 ${upgrade.ToVersion}`);
  await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
 upgrade.ToVersion }).promise();
```

```
  while (true) {
    await usleep(SLEEP_TIME);
    console.log(`Getting the state of canary ${upgrade.Name}`);
    const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
    const state = response.Canary.Status.State;
    console.log(`The state of canary ${upgrade.Name} is ${state}`);
    if (state === 'ERROR' || response.Canary.Status.StateReason) {
      throw response.Canary.Status.StateReason;
    }
    if (state !== 'UPDATING') {
      return;
    }
  }
}

function usleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

使用 Node.js 和 Puppeteer 的运行时版本

使用 Node.js 和 Puppeteer 的第一个运行时版本被命名为 `syn-1.0`。后续运行时版本采用命名约定 `syn-`*`language`*`-`*`majorversion`*`.`*`minorversion`*。从 `syn-nodejs-puppeteer-3.0` 开始，命名约定为 `syn-`*`language`*`-`*`framework`*`-`*`majorversion`*`.`*`minorversion`*

额外添加的 `-beta` 后缀用以显示运行时版本当前为测试预览版。

具有相同主版本号的运行时版本将始终向后兼容。

> **Important**
>
> 以下 CloudWatch Synthetics 运行时系统版本计划于 2024 年 3 月 8 日弃用。
>
> - `syn-nodejs-puppeteer-6.1`
> - `syn-nodejs-puppeteer-6.0`
> - `syn-nodejs-puppeteer-5.1`
> - `syn-nodejs-puppeteer-5.0`
> - `syn-nodejs-puppeteer-4.0`
>
> 有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

> **Important**
>
> 重要提示：在未来的运行时系统版本中，随附的适用于 JavaScript v2 的 AWS SDK 依赖项将被移除，并更新为使用适用于 JavaScript v3 的 AWS SDK。发生这种情况时，您可以更新金丝雀代码引用，也可以继续引用和使用随附的适用于 JavaScript v2 的 AWS SDK 依赖项，方法是将其作为依赖项添加到源代码 zip 文件中。

关于所有运行时版本的说明

使用 `syn-nodejs-puppeteer-3.0` 运行时版本时，请确保金丝雀脚本与 Node.js 12.x 兼容。如果使用早期 `syn-nodejs` 运行时版本，请确保脚本与 Node.js 10.x 兼容。

金丝雀中的 Lambda 代码已配置为具有最多 1 GB 内存。在配置的超时值之后，该超时将适用于每个运行的金丝雀。如果未为金丝雀指定超时值，则 CloudWatch 会根据金丝雀的频率来选择超时值。如要配置超时值，请使此值不要短于 15 秒钟，以预留 Lambda 冷启动以及启动金丝雀工具的时间。

> **Note**
>
> 以下 CloudWatch Synthetics 运行时系统版本于 2024 年 1 月 8 日弃用。这是因为 AWS Lambda 于 2023 年 12 月 4 日弃用了 Lambda Node.js 14 运行时系统。
>
> - `syn-nodejs-puppeteer-3.9`
> - `syn-nodejs-puppeteer-3.8`
> - `syn-nodejs-puppeteer-3.7`
> - `syn-nodejs-puppeteer-3.6`
> - `syn-nodejs-puppeteer-3.5`
>
> 以下 CloudWatch Synthetics 运行时版本于 2022 年 11 月 13 日弃用。这是因为 AWS Lambda 于 2022 年 11 月 14 日弃用 Lambda Node.js 12 运行时。
>
> - `syn-nodejs-puppeteer-3.4`
> - `syn-nodejs-puppeteer-3.3`
> - `syn-nodejs-puppeteer-3.2`
> - `syn-nodejs-puppeteer-3.1`
> - `syn-nodejs-puppeteer-3.0`

有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

syn-nodejs-puppeteer-9.0

`syn-nodejs-puppeteer-9.0` 是 Node.js 和 Puppeteer 最新的 Synthetics 运行时系统。它在 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）均不可用，原因是 Lambda 运行时 Node.js 20.x 在这些区域不可用。

Important

Lambda Node.js 18 和更高版本运行时系统使用适用于 JavaScript V3 的 AWS SDK。如果需要从较早的运行时系统迁移函数，则请关注 GitHub 上的 aws-sdk-js-v3 Migration Workshop。有关 AWS SDK for JavaScript 版本 3 的更多信息，请参阅此博客文章。

主要依赖项：

- Lambda 运行时系统 Node.js 20.x
- Puppeteer-core 版本 22.12.1
- Chromium 版本 126.0.6478.126

错误修复– 修复了有关启用可视化监控功能的错误。

syn-nodejs-puppeteer-8.0

Warning

由于存在错误，`syn-nodejs-puppeteer-8.0` 运行时不支持在金丝雀系统中进行可视化监控。升级到 ??? 以修复可视化监视的错误。

它在 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）均不可用，原因是 Lambda 运行时 Node.js 20.x 在这些区域不可用。

> **Important**
>
> Lambda Node.js 18 和更高版本运行时系统使用适用于 JavaScript V3 的 AWS SDK。如果需要从较早的运行时系统迁移函数，则请关注 GitHub 上的 aws-sdk-js-v3 Migration Workshop。有关 AWS SDK for JavaScript 版本 3 的更多信息，请参阅此博客文章。

主要依赖项：

- Lambda 运行时系统 Node.js 20.x
- Puppeteer-core 22.10.0 版
- Chromium 125.0.6422.112 版

syn-nodejs-puppeteer-8.0 中的新功能：

- 支持双重身份验证
- 某些服务客户端在 Node.js SDK V3 响应中丢失数据的情况的错误修复。

syn-nodejs-puppeteer-7.0

主要依赖项：

- Lambda 运行时系统 Node.js 18.x
- Puppeteer-core 21.9.0 版
- Chromium 121.0.6167.139 版

代码大小：

您可以将大小为 80MB 的代码和依赖项打包到此运行时中。

syn-nodejs-puppeteer-7.0 中的新功能：

- 更新了 Puppeteer 和 Chromium 中捆绑库的版本：Puppeteer 和 Chromium 依赖项已更新到新版本。

> **Important**
>
> 从 Puppeteer 19.7.0 迁移到 Puppeteer 21.9.0 引入了有关测试和筛选条件的重大更改。有关更多信息，请参阅 puppeteer：v20.0.0 和 puppeteer-core：v21.0.0 中的重大更改部分。

推荐升级到 AWS SDK v3

Lambda nodejs18.x 运行时不支持 AWS SDK v2。强烈建议您迁移到 AWS SDK v3。

syn-nodejs-puppeteer-6.2

主要依赖项：

- Lambda 运行时系统 Node.js 18.x
- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-6.2 中的新功能：

- 更新了 Chromium 中捆绑库的版本
- 临时存储监控 — 此运行时系统会在客户账户中添加临时存储监控。
- 错误修复

syn-nodejs-puppeteer-5.2

主要依赖项：

- Lambda 运行时系统 Node.js 16.x
- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-5.2 中的新功能：

- 更新了 Chromium 中捆绑库的版本
- 错误修复

syn-nodejs-puppeteer-6.1

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时系统 Node.js 18.x
- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-6.1 中的新功能：

- 稳定性提高：新增自动重试逻辑，用于处理间歇性 Puppeteer 启动错误。
- 依赖项升级：升级部分第三方依赖项包。
- 没有 Amazon S3 权限的 Canary：通过错误修复使没有任何 Amazon S3 权限的 Canary 仍可运行。这些没有 Amazon S3 权限的 Canary 将无法将屏幕截图或其他构件上传到 Amazon S3。有关 Canary 权限的更多信息，请参阅 金丝雀的必需角色和权限。

> **Important**
>
> 重要提示：在未来的运行时系统版本中，随附的适用于 JavaScript v2 的 AWS SDK 依赖项将被移除，并更新为使用适用于 JavaScript v3 的 AWS SDK。发生这种情况时，您可以更新金丝雀代码引用，也可以继续引用和使用随附的适用于 JavaScript v2 的 AWS SDK 依赖项，方法是将其作为依赖项添加到源代码 zip 文件中。

syn-nodejs-puppeteer-6.0

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时系统 Node.js 18.x
- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-6.0 中的新功能：

- 依赖项升级 – Node.js 依赖项已升级到 18.x。
- 拦截模式支持 – Synthetics 金丝雀运行时系统库中添加了 Puppeteer 合作拦截模式支持。
- 跟踪行为更改 – 将默认跟踪行为更改为了仅跟踪 fetch 和 xhr 请求，而不跟踪资源请求。您可以通过配置 `traceResourceRequests` 选项来启用对资源请求的跟踪。
- 持续时间指标已完善 – `Duration` 指标现在不包括金丝雀用于上传构件、捕获屏幕截图和生成 CloudWatch 指标的操作时间。`Duration` 指标值会报告给 CloudWatch，您也可以在 Synthetics 控制台中进行查看。
- 错误修复 – 清理 Chromium 在金丝雀运行期间崩溃时生成的核心转储。

> **Important**
>
> 重要提示：在未来的运行时系统版本中，随附的适用于 JavaScript v2 的 AWS SDK 依赖项将被移除，并更新为使用适用于 JavaScript v3 的 AWS SDK。发生这种情况时，您可以更新金丝雀代码引用，也可以继续引用和使用随附的适用于 JavaScript v2 的 AWS SDK 依赖项，方法是将其作为依赖项添加到源代码 zip 文件中。

syn-nodejs-puppeteer-5.1

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时系统 Node.js 16.x

- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-5.1 中的错误修复：

- 错误修复：此运行时系统修复了 `syn-nodejs-puppeteer-5.0` 中的一个错误，其中由金丝雀创建的 HAR 文件缺少请求标头。

syn-nodejs-puppeteer-5.0

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时系统 Node.js 16.x
- Puppeteer-core 19.7.0 版
- Chromium 111.0.5563.146 版

syn-nodejs-puppeteer-5.0 中的新功能：

- 依赖关系升级 — Puppeteer-Core 版本已更新至 19.7.0。Chromium 版本已升级至 111.0.5563.146。

> **Important**
>
> 新的 Puppeteer-Core 版本并不完全向后兼容之前版本的 Puppeteer。此版本中的某些更改可能会导致使用已弃用的 Puppeteer 函数的现有金丝雀失败。有关更多信息，请参阅 Puppeteer 变更日志中 Puppeteer-core 19.7.0 至 6.0 版变更日志中的重大更改。

syn-nodejs-puppeteer-4.0

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时系统 Node.js 16.x
- Puppeteer-core 5.5.0 版
- Chromium 92.0.4512 版

syn-nodejs-puppeteer-4.0 中的新功能：

- 依赖项升级 – Node.js 依赖项已更新到 16.x。

已弃用的 Node.js 和 Puppeteer 运行时系统

下列 Node.js 和 Puppeteer 运行时系统已弃用。

syn-nodejs-puppeteer-3.9

> **Important**
>
> 此运行时系统版本已于 2024 年 1 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 14.x
- Puppeteer-core 5.5.0 版
- Chromium 92.0.4512 版

syn-nodejs-puppeteer-3.9 中的新功能：

- 依赖项升级 – 升级一些第三方依赖项包。

syn-nodejs-puppeteer-3.8

> **Important**
>
> 此运行时系统版本已于 2024 年 1 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 14.x
- Puppeteer-core 5.5.0 版
- Chromium 92.0.4512 版

syn-nodejs-puppeteer-3.8 中的新功能：

- 配置文件清理 – 现在，每次金丝雀脚本运行后，都会清理 Chromium 配置文件。

syn-nodejs-puppeteer-3.8 中的错误修复：

- 错误修复 – 以前，在没有屏幕截图的情况下运行后，可视化监控金丝雀脚本有时会停止正常工作。此问题现已修复。

syn-nodejs-puppeteer-3.7

> **Important**
>
> 此运行时系统版本已于 2024 年 1 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 14.x
- Puppeteer-core 5.5.0 版

- Chromium 92.0.4512 版

syn-nodejs-puppeteer-3.7 中的新功能：

- 日志记录增强功能 — 即使 Amazon S3 超时或崩溃，金丝雀也会将日志上传到 Amazon S3。
- Lambda 层大小已减小 — 用于金丝雀的 Lambda 层的大小减小了 34%。

syn-nodejs-puppeteer-3.7 中的错误修复：

- 错误修复 — 日语、简体中文和繁体中文字体将正确呈现。

syn-nodejs-puppeteer-3.6

> **Important**
>
> 此运行时系统版本已于 2024 年 1 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 14.x
- Puppeteer-core 5.5.0 版
- Chromium 92.0.4512 版

syn-nodejs-puppeteer-3.6 中的新功能：

- 更精确的时间戳 — 金丝雀运行的开始时间和停止时间现在精确到毫秒。

syn-nodejs-puppeteer-3.5

> **Important**
>
> 此运行时系统版本已于 2024 年 1 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 14.x
- Puppeteer-core 5.5.0 版
- Chromium 92.0.4512 版

syn-nodejs-puppeteer-3.5 中的新功能：

- 更新的项目依赖项— 此运行时中唯一的新功能是更新的项目依赖项。

syn-nodejs-puppeteer-3.4

> **Important**
>
> 此运行时版本于 2022 年 11 月 13 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 12.x
- Puppeteer-core 5.5.0 版
- Chromium 88.0.4298.0 版

syn-nodejs-puppeteer-3.4 中的新功能:

- 自定义处理程序函数— 您现在可以对金丝雀脚本使用自定义处理程序函数。之前的运行时要求脚本入口点包括 `.handler`。

  您还可以将金丝雀脚本放在任何文件夹中，并将文件夹名称作为处理程序的一部分进行传递。例如，`MyFolder/MyScriptFile.functionname` 可以用作入口点。
- 扩展的 HAR 文件信息— 您现在可以在金丝雀生成的 HAR 文件中看到恶意、待处理和未完成的请求。

syn-nodejs-puppeteer-3.3

> **Important**
>
> 此运行时版本于 2022 年 11 月 13 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 12.x
- Puppeteer-core 5.5.0 版
- Chromium 88.0.4298.0 版

syn-nodejs-puppeteer-3.3 中的新功能：

- 用于构件加密的更多选项 – 对于使用此运行时或更高版本运行时（而不是使用 AWS 托管式密钥）来加密金丝雀存储在 Amazon S3 中的构件的金丝雀，您可以选择使用 AWS KMS 客户托管式密钥或 Amazon S3 托管式密钥。有关更多信息，请参阅 加密金丝雀构件。

syn-nodejs-puppeteer-3.2

> **Important**
>
> 此运行时版本于 2022 年 11 月 13 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 12.x
- Puppeteer-core 5.5.0 版
- Chromium 88.0.4298.0 版

syn-nodejs-puppeteer-3.2 中的新功能：

- 使用屏幕截图进行可视化监控 – 使用此运行时或更高版本运行时的金丝雀可以将运行期间捕获的屏幕截图与相同屏幕截图的基准版本进行比较。如果屏幕截图与指定百分比阈值之间的差异较大，则金丝雀将失败。有关更多信息，请参阅可视化监控或可视监控蓝图。
- 与敏感数据相关的新函数：您可以阻止敏感数据出现在金丝雀日志和报告中。有关更多信息，请参阅 SyntheticsLogHelper 类。
- 已弃用的函数：`RequestResponseLogHelper` 类已弃用，以支持其他新配置选项。有关更多信息，请参阅 RequestResponseLogHelper 类。

syn-nodejs-puppeteer-3.1

> **Important**
>
> 此运行时版本于 2022 年 11 月 13 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 12.x
- Puppeteer-core 5.5.0 版
- Chromium 88.0.4298.0 版

syn-nodejs-puppeteer-3.1 中的新功能：

- CloudWatch 指标配置功能 – 使用此运行时，您可以禁用不需要的指标。否则，金丝雀会发布每次金丝雀运行的各种 CloudWatch 指标。
- 屏幕截图链接 – 您可以在金丝雀步骤完成后将屏幕截图链接到该步骤。为此，您可以使用 takeScreenshot 方法捕获屏幕截图，并使用要与屏幕截图关联的步骤的名称。例如，您可能想要执行某个步骤，添加等待时间，然后捕获屏幕截图。
- 检测信号监控器蓝图可以监控多个 URL – 您可以使用 CloudWatch 控制台中的检测信号监控蓝图来监控多个 URL，并在金丝雀运行报告的步骤摘要中查看每个 URL 的状态、持续时间、关联的屏幕截图和故障原因。

syn-nodejs-puppeteer-3.0

> **Important**
>
> 此运行时版本于 2022 年 11 月 13 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 12.x
- Puppeteer-core 5.5.0 版
- Chromium 88.0.4298.0 版

syn-nodejs-puppeteer-3.0 中的新功能：

- 升级的依赖项 – 此版本使用 Puppeteer 5.5.0 版、Node.js 12.x 和 Chromium 88.0.4298.0。
- 跨区域存储桶访问 – 您现在可以指定另一个区域中的 S3 存储桶作为金丝雀存储其日志文件、屏幕截图和 HAR 文件的存储桶。
- 可用的新函数 – 此版本添加了库函数来检索金丝雀名称和 Synthetics 运行时版本。

  有关更多信息，请参阅 Synthetics 类。

syn-nodejs-2.2

本节包含有关 `syn-nodejs-2.2` 运行时版本的信息。

> **Important**
>
> 此运行时版本已于 2021 年 5 月 28 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 10.x
- Puppeteer-core 3.3.0 版

- Chromium 83.0.4103.0 版

syn-nodejs-2.2 中的新功能：

- 作为 HTTP 步骤监控您的金丝雀 – 您现在可以在一个金丝雀中测试多个 API。每个 API 都作为单独的 HTTP 步骤进行测试，CloudWatch Synthetics 会使用步骤指标和 CloudWatch Synthetics 步骤报告监控每个步骤的状态。CloudWatch Synthetics 会为每个 HTTP 步骤创建 `SuccessPercent` 和 `Duration` 指标。

  此功能由 executeHttpStep(stepName, requestOptions, callback, stepConfig) 函数实现。有关更多信息，请参阅 executeHttpStep(stepName, requestOptions, [callback], [stepConfig])。

  API 金丝雀蓝图已更新为使用此新功能。
- HTTP 请求报告 – 您现在可以查看详细的 HTTP 请求报告，这些报告捕获的是请求/响应标头、响应正文、状态代码、错误和性能计时、TCP 连接时间、TLS 握手时间、第一字节时间和内容传输时间等详细信息。所有使用后台 HTTP/HTTPS 模块的 HTTP 请求都在此处捕获。预设情况下不会捕获标头和响应正文，但可以通过设置配置选项来启用捕获标头和响应正文。
- 全局和步骤级配置 – 您可以在全局级别设置 CloudWatch Synthetics 配置，这些配置应用于金丝雀的所有步骤。您还可以通过传递配置键/值对来启用或禁用某些选项，在步骤级别覆盖这些配置。

  有关更多信息，请参阅 SyntheticsConfiguration 类。
- 继续执行步骤故障配置 – 您可以选择在步骤故障时继续执行金丝雀。对于 `executeHttpStep` 函数，预设情况下，此选项处于启用状态。您可以在全局级别设置此选项一次，或者针对每个步骤做出不同的设置。

syn-nodejs-2.1

> **Important**
>
> 此运行时版本已于 2021 年 5 月 28 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 10.x
- Puppeteer-core 3.3.0 版

- Chromium 83.0.4103.0 版

syn-nodejs-2.1 中的新功能：

- 可配置的屏幕截图行为 – 提供关闭 UI 金丝雀捕获屏幕截图的功能。在使用以前版本运行时的金丝雀中，UI 金丝雀始终在每个步骤前后捕获屏幕截图。在 `syn-nodejs-2.1` 中，可配置此行为。关闭屏幕截图可以降低您的 Amazon S3 存储成本，并有助于您遵守 HIPAA 法规。有关更多信息，请参阅 SyntheticsConfiguration 类。
- 自定义 Google Chrome 启动参数：现在，您可以配置当金丝雀启动 Google Chrome 浏览器窗口时使用的参数。有关更多信息，请参阅 launch(options)。

与更早版本的金丝雀运行时相比，使用 syn-nodejs-2.0 或更高版本时，金丝雀持续时间可能会略有增加。

syn-nodejs-2.0

> **Important**
>
> 此运行时版本已于 2021 年 5 月 28 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 10.x
- Puppeteer-core 3.3.0 版
- Chromium 83.0.4103.0 版

syn-nodejs-2.0 中的新功能：

- 升级的依赖项 – 此运行时版本使用 Puppeteer-core 3.3.0 版和 Chromium 83.0.4103.0 版
- 支持 X-Ray 活动跟踪。金丝雀启用跟踪后，将针对由使用浏览器、AWS SDK 或 HTTP 或 HTTPS 模块的金丝雀发出的所有调用发送 X-Ray 跟踪。启用了跟踪的 Canary 会显示在 X-Ray 跟踪地图中，即使它们没有向启用了跟踪的其他服务或应用程序发送请求。有关更多信息，请参阅 金丝雀和 X-Ray 跟踪。
- Synthetics 报告 – CloudWatch Synthetics 会针对每次金丝雀运行创建一个名为 `SyntheticsReport-PASSED.json` 或 `SyntheticsReport-FAILED.json` 的报告，其中记

录了开始时间、结束时间、状态和故障等数据。报告中还记录金丝雀脚本的每个步骤的 PASSED/FAILED 状态，以及每个步骤捕获的失败和屏幕截图。

- 无效链接检查器报告 – 此运行时中包含的新版本无效链接检查器会创建一个报告，其中包括已检查的链接、状态代码、故障原因（如有）以及源页面和目标页面屏幕截图。
- 新 CloudWatch 指标 – Synthetics 在 `CloudWatchSynthetics` 命名空间中发布名为 `2xx`、`4xx`、`5xx` 和 `RequestFailed` 的指标。这些指标会显示金丝雀运行中 200 秒、400 秒、500 秒和请求故障的数量。在此运行时版本中，这些指标仅针对 UI 金丝雀报告，而不针对 API 金丝雀报告。从 `syn-nodejs-puppeteeer-2.2` 运行时版本开始，也针对 API 金丝雀报告这些指标。
- 可排序的 HAR 文件 – 您现在可以按状态代码、请求大小和持续时间对 HAR 文件进行排序。
- 指标时间戳 – 现在基于 Lambda 调用时间而非金丝雀运行结束时间报告 CloudWatch 指标。

syn-nodejs-2.0 中的错误修复：

- 修复了不报告金丝雀构件上载错误的问题。此类错误现在显示为执行错误。
- 修复了将重定向请求 (3xx) 误记为错误的问题。
- 修复了从 0 开始编号屏幕截图的问题。现在应从 1 开始编号。
- 修复了中文和日文字体屏幕截图乱码的问题。

与更早版本的金丝雀运行时相比，使用 syn-nodejs-2.0 或更高版本时，金丝雀持续时间可能会略有增加。

syn-nodejs-2.0-beta

> **Important**
>
> 此运行时版本已于 2021 年 2 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Lambda 运行时 Node.js 10.x
- Puppeteer-core 3.3.0 版
- Chromium 83.0.4103.0 版

syn-nodejs-2.0-beta 中的新功能：

- 升级的依赖项 – 此运行时版本使用 Puppeteer-core 3.3.0 版和 Chromium 83.0.4103.0 版
- Synthetics 报告 – CloudWatch Synthetics 会针对每次金丝雀运行创建一个名为 `SyntheticsReport-PASSED.json` 或 `SyntheticsReport-FAILED.json` 的报告，其中记录了开始时间、结束时间、状态和故障等数据。报告中还记录金丝雀脚本的每个步骤的 PASSED/FAILED 状态，以及每个步骤捕获的失败和屏幕截图。
- 无效链接检查器报告 – 此运行时中包含的新版本无效链接检查器会创建一个报告，其中包括已检查的链接、状态代码、故障原因（如有）以及源页面和目标页面屏幕截图。
- 新 CloudWatch 指标 – Synthetics 在 `CloudWatchSynthetics` 命名空间中发布名为 `2xx`、`4xx`、`5xx` 和 `RequestFailed` 的指标。这些指标会显示金丝雀运行中 200 秒、400 秒、500 秒和请求故障的数量。仅针对 UI 金丝雀报告而不针对 API 金丝雀报告这些指标。
- 可排序的 HAR 文件 – 您现在可以按状态代码、请求大小和持续时间对 HAR 文件进行排序。
- 指标时间戳 – 现在基于 Lambda 调用时间而非金丝雀运行结束时间报告 CloudWatch 指标。

syn-nodejs-2.0-beta 中的错误修复：

- 修复了不报告金丝雀构件上载错误的问题。此类错误现在显示为执行错误。
- 修复了将重定向请求 (3xx) 误记为错误的问题。
- 修复了从 0 开始编号屏幕截图的问题。现在应从 1 开始编号。
- 修复了中文和日文字体屏幕截图乱码的问题。

syn-1.0

> **Important**
>
> 此运行时版本计划于 2021 年 5 月 28 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

第一个 Synthetics 运行时版本是 `syn-1.0`。

主要依赖项：

- Lambda 运行时 Node.js 10.x

- Puppeteer-core 1.14.0 版
- 与 Puppeteer-core 1.14.0 匹配的 Chromium 版本

使用 Python 和 Selenium Webdriver 的运行时版本

以下各节包含有关用于 Python 和 Selenium Webdriver 的 CloudWatch Synthetics 运行时版本的信息。Selenium 是一种开源浏览器自动化工具。有关 Selenium 的更多信息，请参阅 www.selenium.dev/

这些运行时版本的命名约定为 `syn-language-framework-majorversion.minorversion`。

> **Important**
>
> 以下 CloudWatch Synthetics 运行时系统版本计划于 2024 年 3 月 8 日弃用。
>
> - `syn-python-selenium-2.0`
> - `syn-python-selenium-1.3`
> - `syn-python-selenium-1.2`
> - `syn-python-selenium-1.1`
> - `syn-python-selenium-1.0`
>
> 有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

syn-python-selenium-4.0

版本 4.0 是适用于 Python 和 Selenium 的最新 CloudWatch Synthetics 运行时。

主要依赖项：

- Python 3.9
- Selenium 4.15.1
- Chromium 版本 126.0.6478.126

syn-python-selenium-4.0 中的新功能：

- 修复了 HAR 解析器日志记录中的错误。

syn-python-selenium-3.0

主要依赖项：

- Python 3.8
- Selenium 4.15.1
- Chromium 121.0.6167.139 版

syn-python-selenium-3.0 中的新功能：

- 更新了 Chromium 中捆绑库的版本：Chromium 依赖项已更新到新版本。

syn-python-selenium-2.1

主要依赖项：

- Python 3.8
- Selenium 4.15.1
- Chromium 111.0.5563.146 版

syn-python-selenium-2.1 中的新功能：

- 更新了 Chromium 中捆绑库的版本：Chromium 和 Selenium 依赖项已更新到新版本。

syn-python-selenium-2.0

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Python 3.8
- Selenium 4.10.0
- Chromium 111.0.5563.146 版

syn-python-selenium-2.0 中的新功能：

- 更新了依赖项 — Chromium 和 Selenium 依赖项已更新到新版本。

syn-python-selenium-2.0 中的错误修复：

- 添加了时间戳 — 已将时间戳添加到金丝雀日志。
- 会话重用 — 修复了一个错误，因此金丝雀现在无法重用以前金丝雀运行的会话。

syn-python-selenium-1.3

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Python 3.8
- Selenium 3.141.0
- Chromium 92.0.4512.0 版

syn-python-selenium-1.3 中的新功能：

- 更精确的时间戳 — 金丝雀运行的开始时间和停止时间现在精确到毫秒。

syn-python-selenium-1.2

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Python 3.8
- Selenium 3.141.0
- Chromium 92.0.4512.0 版

- 更新的项目依赖项— 此运行时中唯一的新功能是更新的项目依赖项。

syn-python-selenium-1.1

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Python 3.8
- Selenium 3.141.0
- Chromium 83.0.4103.0 版

功能：

- 自定义处理程序函数— 您现在可以对金丝雀脚本使用自定义处理程序函数。之前的运行时要求脚本入口点包括 `.handler`。

  您还可以将金丝雀脚本放在任何文件夹中，并将文件夹名称作为处理程序的一部分进行传递。例如，`MyFolder/MyScriptFile.functionname` 可以用作入口点。
- 用于添加指标和步骤失败配置的配置选项— 这些选项已在适用于 Node.js 金丝雀的运行时中提供。有关更多信息，请参阅 SyntheticsConfiguration 类。
- Chrome 中的自定义参数— 您现在可以用无痕模式打开浏览器或用代理服务器配置进行传递。有关更多信息，请参阅 Chrome()。
- 跨区域构件存储桶— 金丝雀可以将其构件存储在不同区域的 Simple Storage Service (Amazon S3) 存储桶中。
- 错误修复，包括修复 **`index.py`** 问题— 对于之前的运行时，名为 `index.py` 的金丝雀文件导致异常，因为其与库文件的名称冲突。此问题现已修复。

syn-python-selenium-1.0

> **Important**
>
> 此运行时系统版本计划于 2024 年 3 月 8 日弃用。有关更多信息，请参阅 CloudWatch Synthetics 运行时支持策略。

主要依赖项：

- Python 3.8
- Selenium 3.141.0
- Chromium 83.0.4103.0 版

功能：

- Selenium 支持 – 您可以使用 Selenium 测试框架编写金丝雀脚本。您可以将其他地方的 Selenium 脚本添加到 CloudWatch Synthetics 中，只需做出最少的更改，它们即可用于 AWS 服务。

编写金丝雀脚本

以下部分介绍了如何编写 Canary 脚本，以及如何将 Canary 与其他 AWS 服务及外部依赖项和库集成。

主题

- 编写 Node.js 金丝雀脚本
- 编写 Python 金丝雀脚本
- 更改现有 Puppeteer 脚本以使用 Synthetics 金丝雀
- 更改现有的 Puppeteer Synthetics 脚本以验证非标准证书

编写 Node.js 金丝雀脚本

主题

- 从 Scratch 创建 CloudWatch Synthetics 金丝雀
- 将 Node.js 金丝雀文件打包

- 更改现有 Puppeteer 脚本以将其用作 Synthetics 金丝雀
- 环境变量
- 将您的金丝雀与其他 AWS 服务集成
- 强制金丝雀使用静态 IP 地址

从 Scratch 创建 CloudWatch Synthetics 金丝雀

这里是一个极简 Synthetics 金丝雀脚本示例。此脚本将成功通过一次运行，并返回一个字符串。要查看失败的金丝雀示例，请将 `let fail = false;` 更改为 `let fail = true;`。

您必须为金丝雀脚本定义入口点函数。要查看如何将文件上载到指定作为金丝雀 `ArtifactS3Location` 的 Amazon S3 位置，请在 /tmp 文件夹下创建这些文件。脚本运行后，将“通过/失败”状态和持续时间指标发布到 CloudWatch，并将 /tmp 下的文件上载到 S3。

```
const basicCustomEntryPoint = async function () {

    // Insert your code here

    // Perform multi-step pass/fail check

    // Log decisions made and results to /tmp

    // Be sure to wait for all your code paths to complete
    // before returning control back to Synthetics.
    // In that way, your canary will not finish and report success
    // before your code has finished executing

    // Throw to fail, return to succeed
    let fail = false;
    if (fail) {
        throw "Failed basicCanary check.";
    }

    return "Successfully completed basicCanary checks.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

接下来，我们将扩展脚本以使用 Synthetics 日志记录并使用 AWS SDK 进行调用。出于演示目的，此脚本将创建 Amazon DynamoDB 客户端并调用 DynamoDB listTables API。它会记录对请求的响应，并根据请求是否成功来记录通过还是失败。

```
const log = require('SyntheticsLogger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

    log.info("Starting DynamoDB:listTables canary.");

    let dynamodb = new AWS.DynamoDB();
    var params = {};
    let request = await dynamodb.listTables(params);
    try {
        let response = await request.promise();
        log.info("listTables response: " + JSON.stringify(response));
    } catch (err) {
        log.error("listTables error: " + JSON.stringify(err), err.stack);
        throw err;
    }

    return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

将 Node.js 金丝雀文件打包

如果您使用 Simple Storage Service (Amazon S3) 位置上传金丝雀脚本，则 zip 文件应在此文件夹结构下包含脚本: nodejs/node_modules/*myCanaryFilename.js file*。

如果您有多个 .js 文件，或者您的脚本依赖于某个依赖项，则您可以将它们打包到包含文件夹结构 nodejs/node_modules/*myCanaryFilename.js file and other folders and files* 的单个 ZIP 文件中。如果您使用 syn-nodejs-puppeteer-3.4 或者之后版本，您可以选择将金丝雀文件放在另一个文件夹中，然后创建与此类似的文件夹结构: nodejs/

node_modules/*myFolder*/*myCanaryFilename.js file and other folders and files*。

处理程序名称

请务必将您的金丝雀脚本入口点（处理程序）设置为 myCanaryFilename.functionName，以匹配脚本入口点的文件名。如果您使用 syn-nodejs-puppeteer-3.4 之前版本的运行时，则 functionName 必须是 handler。如果您使用 syn-nodejs-puppeteer-3.4 或之后版本，您可以选择任何函数名称作为处理程序。如果您使用 syn-nodejs-puppeteer-3.4 或之后版本，您也可以选择将金丝雀存储在单独的文件夹中，例如 nodejs/node_modules/myFolder/my_canary_filename。如果将其存储在单独的文件夹中，请在脚本入口点中指定该路径，例如 myFolder/my_canary_filename.functionName。

更改现有 Puppeteer 脚本以将其用作 Synthetics 金丝雀

本节介绍如何对 Puppeteer 脚本进行修改，以将其作为 Synthetics 金丝雀脚本运行。有关 Puppeteer 的更多信息，请参阅 Puppeteer API v1.14.0。

我们从这个示例 Puppeteer 开始：

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

转换步骤如下：

- 创建和导出 handler 函数。处理程序是脚本的入口点函数。如果您使用 syn-nodejs-puppeteer-3.4 之前版本的运行时，处理程序函数必须命名为 handler。如果您使用 syn-nodejs-puppeteer-3.4 或之后版本，函数可以有任何名称，但必须与脚本中使用的名称相同。此外，如果您使用 syn-nodejs-puppeteer-3.4 或之后版本，您可以将脚本存储在任何文件夹下，并将该文件夹指定为处理程序名称的一部分。

  ```
  const basicPuppeteerExample = async function () {};
  ```

```
exports.handler = async () => {
    return await basicPuppeteerExample();
};
```

- 使用 Synthetics 依赖项。

```
var synthetics = require('Synthetics');
```

- 使用 Synthetics.getPage 函数获取 Puppeteer Page 对象。

```
const page = await synthetics.getPage();
```

Synthetics.getPage 函数返回的页面对象指示需要记录 page.on request、response 和 requestfailed 事件。Synthetics 还为页面上的请求和响应设置 HAR 文件生成，并将金丝雀 ARN 添加到页面上的传出请求的 user-agent 标头。

该脚本现已准备好作为 Synthetics 金丝雀运行。更新的脚本如下：

```
var synthetics = require('Synthetics');  // Synthetics dependency

const basicPuppeteerExample = async function () {
    const page = await synthetics.getPage(); // Get instrumented page from Synthetics
    await page.goto('https://example.com');
    await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp
 folder
};

exports.handler = async () => {  // Exported handler function
    return await basicPuppeteerExample();
};
```

环境变量

您可以在创建金丝雀时使用环境变量。这样，您就能够编写单个金丝雀脚本，然后将该脚本与不同的值相结合，快速创建具有类似任务的多个金丝雀。

例如，假定您的企业具有用于不同软件开发阶段的 prod、dev 和 pre-release 端点，而您需要创建金丝雀来测试其中每个端点。您可以编写一个测试软件的金丝雀脚本，然后在分别创建三个金丝雀时为端点环境变量指定不同的值。之后，在创建金丝雀时，您可以指定要用于环境变量的脚本和值。

环境变量的名称可以包含字母、数字和下划线字符，必须以字母开头，并且至少为两个字符。环境变量的总大小不能超过 4 KB。不能将任何 Lambda 预留环境变量指定为环境变量的名称。有关预留环境变量的更多信息，请参阅运行时环境变量。

> **Important**
>
> 环境变量键和值未加密。请勿在其中存储敏感信息。

下面的示例脚本使用两个环境变量。此脚本用于检查网页是否可用的金丝雀。该金丝雀使用环境变量来参数化所检查的 URL 和所使用的 CloudWatch Synthetics 日志级别。

以下函数将 `LogLevel` 设置为 `LOG_LEVEL` 环境变量的值。

```
synthetics.setLogLevel(process.env.LOG_LEVEL);
```

此函数将 URL 设置为 URL 环境变量的值。

```
const URL = process.env.URL;
```

以下是完整的脚本。当您使用此脚本创建金丝雀时，您可以指定 `LOG_LEVEL` 和 URL 环境变量的值。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadEnvironmentVariable = async function () {

    // Setting the log level (0-3)
    synthetics.setLogLevel(process.env.LOG_LEVEL);
    // INSERT URL here
    const URL = process.env.URL;

    let page = await synthetics.getPage();
    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }
    //Wait for page to render.
```

```
    //Increase or decrease wait time based on endpoint being monitored.
    await page.waitFor(15000);
    await synthetics.takeScreenshot('loaded', 'loaded');
    let pageTitle = await page.title();
    log.info('Page title: ' + pageTitle);
    log.debug('Environment variable:' + process.env.URL);

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadEnvironmentVariable();
};
```

将环境变量传递到脚本

要在控制台中创建金丝雀时将环境变量传递给脚本，请在控制台上的 Environment variables（环境变量）部分指定环境变量的密钥和值。有关更多信息，请参阅 创建金丝雀。

要通过 API 或 AWS CLI 传递环境变量，请使用 RunConfig 部分中的 EnvironmentVariables 参数。以下为 AWS CLI 命令示例，该命令创建一个使用两个环境变量（具有 Environment 和 Region 密钥）的金丝雀。

```
aws synthetics create-canary --cli-input-json '{
   "Name":"nameofCanary",
   "ExecutionRoleArn":"roleArn",
   "ArtifactS3Location":"s3://cw-syn-results-123456789012-us-west-2",
   "Schedule":{
      "Expression":"rate(0 minute)",
      "DurationInSeconds":604800
   },
   "Code":{
      "S3Bucket": "canarycreation",
      "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
      "Handler":"pageLoadBlueprint.handler"
   },
   "RunConfig": {
      "TimeoutInSeconds":60,
      "EnvironmentVariables": {
```

```
         "Environment":"Production",
         "Region": "us-west-1"
      }
   },
   "SuccessRetentionPeriodInDays":13,
   "FailureRetentionPeriodInDays":13,
   "RuntimeVersion":"syn-nodejs-2.0"
}'
```

将您的金丝雀与其他 AWS 服务集成

所有金丝雀都可以使用 AWS SDK 库。在编写金丝雀时，您可以使用此库将金丝雀与其他 AWS 服务集成。

为此，您需要将以下代码添加到您的金丝雀中。对于这些例子，AWS Secrets Manager 用作与金丝雀集成的服务。

- 导入 AWS SDK。

```
const AWS = require('aws-sdk');
```

- 为要集成的 AWS 服务创建客户端。

```
const secretsManager = new AWS.SecretsManager();
```

- 使用客户端对该服务进行 API 调用。

```
var params = {
  SecretId: secretName
};
return await secretsManager.getSecretValue(params).promise();
```

下面的金丝雀脚本代码段更详细地演示了与 Secrets Manager 集成的示例。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager();

const getSecrets = async (secretName) => {
```

```
    var params = {
        SecretId: secretName
    };
    return await secretsManager.getSecretValue(params).promise();
}

const secretsExample = async function () {
    let URL = "<URL>";
    let page = await synthetics.getPage();

    log.info(`Navigating to URL: ${URL}`);
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});

    // Fetch secrets
    let secrets = await getSecrets("secretname")

    /**
    * Use secrets to login.
    *
    * Assuming secrets are stored in a JSON format like:
    * {
    *   "username": "<USERNAME>",
    *   "password": "<PASSWORD>"
    * }
    **/
    let secretsObj = JSON.parse(secrets.SecretString);
    await synthetics.executeStep('login', async function () {
        await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
        await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

        await Promise.all([
          page.waitForNavigation({ timeout: 30000 }),
          await page.click(">SUBMIT-BUTTON-SELECTOR<")
        ]);
    });

    // Verify login was successful
    await synthetics.executeStep('verify', async function () {
        await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
    });
};

exports.handler = async () => {
```

```
    return await secretsExample();
};
```

强制金丝雀使用静态 IP 地址

您可以设置金丝雀，使其使用静态 IP 地址。

强制金丝雀使用静态 IP 地址

1. 创建新的 VPC。有关更多信息，请参阅在您的 VPC 中使用 DNS。
2. 创建新的互联网网关。有关更多信息，请参阅在您的 VPC 中添加互联网网关。
3. 在新的 VPC 内创建公有子网。
4. 向 VPC 添加新的路由表。
5. 在新路由表中添加一条从 `0.0.0.0/0` 到互联网网关的路由。
6. 将新的路由表与公有子网关联。
7. 创建弹性 IP 地址。有关更多信息，请参阅弹性 IP 地址。
8. 创建一个新的 NAT 网关，并将其分配给公有子网和弹性 IP 地址。
9. 在 VPC 中创建私有子网。
10. 向 VPC 默认路由表中添加一条从 `0.0.0.0/0` 到 NAT 网关的路由
11. 创建金丝雀。

编写 Python 金丝雀脚本

此脚本将成功通过一次运行，并返回一个字符串。要查看失败的金丝雀示例，请将 fail = False 更改为 fail = True

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
```

```
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

将 Python 金丝雀文件打包

如果您有多个 .py 文件或脚本具有依赖项，则您可以将它们捆绑到单个 ZIP 格式文件中。如果您使用 syn-python-selenium-1.1 运行时，此 ZIP 格式文件必须将主金丝雀 .py 文件包含在 python 文件夹中，例如 python/my_canary_filename.py。如果您使用 syn-python-selenium-1.1 或之后版本，您可以选择使用其他文件夹，例如 python/myFolder/my_canary_filename.py。

此 ZIP 格式文件应包含所有必要的文件夹和文件，但其他文件不需要位于 python 文件夹中。

请务必将您的金丝雀脚本入口点设置为 my_canary_filename.functionName，以匹配脚本入口点的文件名和函数名称。如果您使用 syn-python-selenium-1.0 运行时，则 functionName 必须是 handler。如果您使用 syn-python-selenium-1.1 或之后版本，该处理程序名称限制不适用，您也可以选择将金丝雀存储在单独的文件夹中，例如 python/myFolder/my_canary_filename.py。如果将其存储在单独的文件夹中，请在脚本入口点中指定该路径，例如 myFolder/my_canary_filename.functionName。

更改现有 Puppeteer 脚本以使用 Synthetics 金丝雀

您可以快速修改用于 Python 和 Selenium 的现有脚本以用作金丝雀。有关 Selenium 的更多信息，请参阅 www.selenium.dev/。

在本示例中，将从以下 Selenium 脚本开始：

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

转换步骤如下。

转换 Selenium 脚本以用作金丝雀

1. 更改 import 语句以使用 aws_synthetics 模块中的 Selenium：

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
```

`aws_synthetics` 模块中的 Selenium 确保金丝雀可以发出指标和日志、生成 HAR 文件及使用其他 CloudWatch Synthetics 功能。

2. 创建一个处理程序函数并调用您的 Selenium 方法。处理程序是脚本的入口点函数。

   如果您使用 `syn-python-selenium-1.0`，处理程序函数必须命名为 `handler`。如果您使用 `syn-python-selenium-1.1` 或之后版本，函数可以有任何名称，但必须与脚本中使用的名称相同。此外，如果您使用 `syn-python-selenium-1.1` 或之后版本，您可以将脚本存储在任何文件夹下，并将该文件夹指定为处理程序名称的一部分。

```
def handler(event, context):
    basic_selenium_script()
```

脚本现已更新为 CloudWatch Synthetics 金丝雀。更新的脚本如下：

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

更改现有的 Puppeteer Synthetics 脚本以验证非标准证书

Synthetics Canary 的一个重要用例是让您监测自己的端点。如果您想监测尚未准备好接收外部流量的端点，这种监测有时可能意味着您尚未拥有由可信的第三方证书颁发机构签署的正确证书。

对于这种情况，有如下两种可能的解决方案：

- 要对客户端证书进行身份验证，请参阅 How to validate authentication using Amazon CloudWatch Synthetics – Part 2。
- 要对自签名证书进行身份验证，请参阅 How to validate authentication with self-signed certificates in Amazon CloudWatch Synthetics

使用 CloudWatch Synthetics Canary 时，您的选择并不局限于以上两种。您可以通过扩展 Canary 代码来扩展这些功能并添加业务逻辑。

> **Note**
>
> 在 Python 运行时上运行的 Synthetics Canary 本身就启用了 `--ignore-certificate-errors` 标志，因此这些 Canary 在访问具有非标准证书配置的网站时，不应出现任何问题。

可用于金丝雀脚本的库函数

CloudWatch Synthetics 中包含多个您在编写用作金丝雀的 Node.js 脚本时可以调用的内置类和函数。

某些类和函数适用于 UI 和 API 金丝雀。其他函数仅适用于 UI 金丝雀。UI 金丝雀是使用 `getPage()` 函数，并使用 Puppeteer 作为 Web 驱动程序进行导航和与网页交互的金丝雀。

> **Note**
>
> 每当您升级金丝雀以使用新版本的 Synthetics 运行时时，您的金丝雀使用的所有 Synthetics 库函数也会自动升级到 Synthetics 运行时支持的相同 NodeJS 版本。

主题

- 可用于 Node.js 金丝雀脚本的库函数
- 可用于使用 Selenium 的 Python 金丝雀脚本的库函数

可用于 Node.js 金丝雀脚本的库函数

本节列出了可用于 Node.js 金丝雀脚本的库函数。

主题

- 适用于所有金丝雀的 Node.js 库类和函数
- 仅适用于 UI 金丝雀的 Node.js 库类和函数
- 仅适用于 API 金丝雀的 Node.js 库类和函数

适用于所有金丝雀的 Node.js 库类和函数

以下用于 Node.js 的 CloudWatch Synthetics 库函数适用于所有金丝雀。

主题

- Synthetics 类
- SyntheticsConfiguration 类
- Synthetics Logger
- SyntheticsLogHelper 类

Synthetics 类

以下适用于所有金丝雀的函数都属于 Synthetics 类。

addExecutionError(errorMessage, ex);

`errorMessage` 描述错误，`ex` 指遇到的异常

您可以使用 `addExecutionError` 来设置金丝雀的执行错误。它会在不中断脚本执行的情况下导致金丝雀失败。它也不会影响 `successPercent` 指标。

只有当错误对于指示金丝雀脚本成功或故障并不重要时，才应将错误作为执行错误进行跟踪。

以下是使用 `addExecutionError` 的示例。您将会监控端点的可用性，并在页面加载完成后捕获屏幕截图。由于捕获屏幕截图故障并不会决定端点的可用性，因此您可以捕获在截图时遇到的任何错误，并将它们添加为执行错误。可用性指标仍会指示端点已启动并正在运行，但金丝雀状态会被标记为失败。以下示例代码块将会捕获此类错误并将其添加为执行错误。

```
try {
        await synthetics.takeScreenshot(stepName, "loaded");
    } catch(ex) {
        synthetics.addExecutionError('Unable to take screenshot ', ex);
    }
```

getCanaryName();

返回金丝雀的名称。

getCanaryArn();

返回金丝雀的 ARN。

getCanaryUserAgentString();

返回金丝雀的自定义用户代理。

getRuntimeVersion();

此函数在 syn-nodejs-puppeteer-3.0 和更高版本的运行时中可用。其返回金丝雀的 Synthetics 运行时版本。例如，返回值可能为 syn-nodejs-puppeteer-3.0。

getLogLevel();

检索 Synthetics 库的当前日志级别。可能的值包括：

- 0 – 调试
- 1 – 信息
- 2 – 警告
- 3 – 错误

例如：

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

设置 Synthetics 库的日志级别。可能的值包括：

- 0 – 调试
- 1 – 信息
- 2 – 警告
- 3 – 错误

例如：

```
synthetics.setLogLevel(0);
```

SyntheticsConfiguration 类

此类仅在 syn-nodejs-2.1 或更高版本的运行时中可用。

您可以使用 SyntheticsConfiguration 类来配置 Synthetics 库函数的行为。例如，您可以使用此类将 executeStep() 函数配置为不捕获屏幕截图。

您可以在全局级别设置 CloudWatch Synthetics 配置，这些配置会应用于金丝雀的所有步骤。您还可以通过传递配置键/值对，在步骤级别覆盖这些配置。

您可以在步骤级别传入选项。有关示例，请参阅 async executeStep(stepName, functionToExecute, [stepConfig]); 和 executeHttpStep(stepName, requestOptions, [callback], [stepConfig])。

函数定义：

setConfig(options)

options 是一个对象，它是一组适用于您的金丝雀的可配置选项。以下各节将介绍 *options* 中的可能字段。

所有金丝雀的 setConfig(options)

对于使用 syn-nodejs-puppeteer-3.2 或更高版本的金丝雀，setConfig 的 (options) 可包含以下参数：

- includeRequestHeaders（布尔值）– 是否在报告中包含请求标头。默认为 false。
- includeResponseHeaders（布尔值）– 是否在报告中包含响应标头。默认为 false。
- restrictedHeaders（数组）– 要忽略的标头值列表（如果包含标头）。这对请求标头和响应标头均适用。例如，您可以将 includeRequestHeaders 传递为 true 并将 restrictedHeaders 传递为 ['Authorization']，来隐藏您的凭证。
- includeRequestBody（布尔值）– 是否在报告中包含请求体。默认为 false。
- includeResponseBody（布尔值）– 是否在报告中包含响应体。默认为 false。

  如果您启用了 includeResponseBody 或 logResponseBody，则某些 API（例如 aws-sdk v3 客户端）的响应中不会返回该数据对象。这是因为 Node.js 和所用响应对象类型的限制。

CloudWatch 指标的相关 setConfig(options)

对于使用 syn-nodejs-puppeteer-3.1 或更高版本的金丝雀，setConfig 的 (options) 可包含以下布尔参数，这些参数决定将由金丝雀发布的指标。其中每个选项的默认值均为 true。以 aggregated 开头的选项决定是否不带 CanaryName 维度发出指标。您可以使用该等指标查看所有金丝雀的聚合结果。其他选项决定是否带 CanaryName 维度发出指标。您可以使用该等指标查看每个金丝雀的结果。

有关金丝雀发出的 CloudWatch 指标的列表，请参阅 金丝雀发布的 CloudWatch 指标。

- failedCanaryMetric（布尔值）– 是否发出此金丝雀的 Failed 指标（带 CanaryName 维度）。默认为 true。

- failedRequestsMetric（布尔值）– 是否发出此金丝雀的 Failed requests 指标（带 CanaryName 维度）。默认为 true。
- _2xxMetric（布尔值）– 是否发出此金丝雀的 2xx 指标（带 CanaryName 维度）。默认为 true。
- _4xxMetric（布尔值）– 是否发出此金丝雀的 4xx 指标（带 CanaryName 维度）。默认为 true。
- _5xxMetric（布尔值）– 是否发出此金丝雀的 5xx 指标（带 CanaryName 维度）。默认为 true。
- stepDurationMetric（布尔值）– 是否发出此金丝雀的 Step duration 指标（带 CanaryName StepName 维度）。默认为 true。
- stepSuccessMetric（布尔值）– 是否发出此金丝雀的 Step success 指标（带 CanaryName StepName 维度）。默认为 true。
- aggregatedFailedCanaryMetric（布尔值）– 是否发出此金丝雀的 Failed 指标（不带 CanaryName 维度）。默认为 true。
- aggregatedFailedRequestsMetric（布尔值）– 是否发出此金丝雀的 Failed Requests 指标（不带 CanaryName 维度）。默认为 true。
- aggregated2xxMetric（布尔值）– 是否发出此金丝雀的 2xx 指标（不带 CanaryName 维度）。默认为 true。
- aggregated4xxMetric（布尔值）– 是否发出此金丝雀的 4xx 指标（不带 CanaryName 维度）。默认为 true。
- aggregated5xxMetric（布尔值）– 是否发出此金丝雀的 5xx 指标（不带 CanaryName 维度）。默认为 true。
- visualMonitoringSuccessPercentMetric（布尔值）– 是否发出此金丝雀的 visualMonitoringSuccessPercent 指标。默认为 true。
- visualMonitoringTotalComparisonsMetric（布尔值）– 是否发出此金丝雀的 visualMonitoringTotalComparisons 指标。默认为 false。
- stepsReport（布尔值）– 是否报告步骤执行摘要。默认为 true。
- includeUrlPassword（布尔值）– 是否包含 URL 中显示的密码。预设情况下，URL 中显示的密码会在日志和报告中进行编辑，以防泄露敏感数据。默认为 false。
- restrictedUrlParameters（数组）– 要编辑的 URL 路径或查询参数的列表。这适用于出现在日志、报告和错误中的 URL。此参数区分大小写。您可以传递星号 (*) 作为值来编辑所有 URL 路径和查询参数值。默认值为空数组。

- logRequest（布尔值）– 是否将每个请求都记录在金丝雀日志中。对于 UI 金丝雀，这会记录浏览器发送的每个请求。默认为 true。
- logResponse（布尔值）– 是否将每个响应都记录在金丝雀日志中。对于 UI 金丝雀，这会记录浏览器收到的每个响应。默认为 true。
- logRequestBody（布尔值）– 是否随请求一起将请求体记录在金丝雀日志中。仅当 logRequest 为 true 时，此配置才适用。默认为 false。
- logResponseBody（布尔值）– 是否随响应一起将响应体记录在金丝雀日志中。仅当 logResponse 为 true 时，此配置才适用。默认为 false。

  如果您启用了 includeResponseBody 或 logResponseBody，则某些 API（例如 aws-sdk v3 客户端）的响应中不会返回该数据对象。这是因为 Node.js 和所用响应对象类型的限制。
- logRequestHeaders（布尔值）– 是否随请求一起将请求标头记录在金丝雀日志中。仅当 logRequest 为 true 时，此配置才适用。默认为 false。

  请注意，includeRequestHeaders 会在构件中启用标头。
- logResponseHeaders（布尔值）– 是否随响应一起将响应标头记录在金丝雀日志中。仅当 logResponse 为 true 时，此配置才适用。默认为 false。

  请注意，includeResponseHeaders 会在构件中启用标头。

> **Note**
>
> 始终会发出每个金丝雀的 Duration 和 SuccessPercent 指标，这两个指标都会带有和不带 CanaryName 指标。

用于启用或禁用指标的方法

disableAggregatedRequestMetrics()

禁用金丝雀发出以不带 CanaryName 维度形式发出的所有请求指标。

disableRequestMetrics()

禁用所有请求指标，包括每个金丝雀的指标和跨所有金丝雀聚合的指标。

disableStepMetrics()

禁用所有步骤指标，包括步骤成功指标和步骤持续时间指标。

enableAggregatedRequestMetrics()

启用金丝雀发出以不带 `CanaryName` 维度形式发出的所有请求指标。

enableRequestMetrics()

启用所有请求指标，包括每个金丝雀的指标和跨所有金丝雀聚合的指标。

enableStepMetrics()

启用所有步骤指标，包括步骤成功指标和步骤持续时间指标。

get2xxMetric()

返回金丝雀是否发出带 `CanaryName` 维度的 `2xx` 指标。

get4xxMetric()

返回金丝雀是否发出带 `CanaryName` 维度的 `4xx` 指标。

get5xxMetric()

返回金丝雀是否发出带 `CanaryName` 维度的 `5xx` 指标。

getAggregated2xxMetric()

返回金丝雀是否发出不带维度的 `2xx` 指标。

getAggregated4xxMetric()

返回金丝雀是否发出不带维度的 `4xx` 指标。

getAggregatedFailedCanaryMetric()

返回金丝雀是否发出不带维度的 `Failed` 指标。

getAggregatedFailedRequestsMetric()

返回金丝雀是否发出不带维度的 `Failed requests` 指标。

getAggregated5xxMetric()

返回金丝雀是否发出不带维度的 `5xx` 指标。

getFailedCanaryMetric()

返回金丝雀是否发出带 CanaryName 维度的 Failed 指标。

getFailedRequestsMetric()

返回金丝雀是否发出带 CanaryName 维度的 Failed requests 指标。

getStepDurationMetric()

返回金丝雀是否发出带有此金丝雀的 CanaryName 维度的 Duration 指标。

getStepSuccessMetric()

返回金丝雀是否发出带有此金丝雀的 CanaryName 维度的 StepSuccess 指标。

with2xxMetric(_2xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 2xx 指标。

with4xxMetric(_4xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 4xx 指标。

with5xxMetric(_5xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 5xx 指标。

withAggregated2xxMetric(aggregated2xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 2xx 指标。

withAggregated4xxMetric(aggregated4xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 4xx 指标。

withAggregated5xxMetric(aggregated5xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 5xx 指标。

withAggregatedFailedCanaryMetric(aggregatedFailedCanaryMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 Failed 指标。

withAggregatedFailedRequestsMetric(aggregatedFailedRequestsMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 Failed requests 指标。

withFailedCanaryMetric(failedCanaryMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Failed 指标。

withFailedRequestsMetric(failedRequestsMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Failed requests 指标。

withStepDurationMetric(stepDurationMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Duration 指标。

withStepSuccessMetric(stepSuccessMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 StepSuccess 指标。

启用或禁用其他功能的方法

withHarFile()

接受一个布尔参数，该参数指定是否为此金丝雀创建 HAR 文件。

withStepsReport()

接受一个布尔参数，该参数指定是否报告此金丝雀的步骤执行摘要。

withIncludeUrlPassword()

接受一个布尔参数，该参数指定是否在日志和报告中包含 URL 中出现的密码。

withRestrictedUrlParameters()

接受要编辑的 URL 路径或查询参数的数组。这适用于出现在日志、报告和错误中的 URL。您可以传递星号 (*) 作为值来编辑所有 URL 路径和查询参数值

withLogRequest()

接受一个布尔参数，该参数指定是否将每个请求记录在金丝雀的日志中。

withLogResponse()

接受一个布尔参数，该参数指定是否将每个响应记录在金丝雀的日志中。

withLogRequestBody()

接受一个布尔参数，该参数指定是否将每个请求体记录在金丝雀的日志中。

withLogResponseBody()

接受一个布尔参数，该参数指定是否将每个响应体记录在金丝雀的日志中。

withLogRequestHeaders()

接受一个布尔参数，该参数指定是否将每个请求标头记录在金丝雀的日志中。

withLogResponseHeaders()

接受一个布尔参数，该参数指定是否将每个响应标头记录在金丝雀日志中。

getHarFile()

返回金丝雀是否创建 HAR 文件。

getStepsReport()

返回金丝雀是否报告步骤执行摘要。

getIncludeUrlPassword()

返回金丝雀是否在日志和报告中包含 URL 中出现的密码。

getRestrictedUrlParameters()

返回金丝雀是否编辑 URL 路径或查询参数。

getLogRequest()

返回金丝雀是否将每个请求记录在金丝雀的日志中。

getLogResponse()

返回金丝雀是否将每个响应记录在金丝雀的日志中。

getLogRequestBody()

返回金丝雀是否将每个请求体记录在金丝雀的日志中。

getLogResponseBody()

返回金丝雀是否将每个响应体记录在金丝雀的日志中。

getLogRequestHeaders()

返回金丝雀是否将每个请求标头记录在金丝雀的日志中。

getLogResponseHeaders()

返回金丝雀是否将每个响应标头记录在金丝雀的日志中。

所有金丝雀的函数

- withIncludeRequestHeaders(includeRequestHeaders)
- withIncludeResponseHeaders（包括响应标题）
- withRestrictedHeaders(restrictedHeaders)
- withIncludeRequestBody(includeRequestBody)
- withIncludeResponseBody(includeResponseBody)
- enableReportingOptions() – 启用所有报告选项-- includeRequestHeaders、includeResponseHeaders、includeRequestBody 和 includeResponseBody。
- disableReportingOptions() – 禁用所有报告选项-- includeRequestHeaders、includeResponseHeaders、includeRequestBody 和 includeResponseBody。

用于 UI 金丝雀的 setConfig(options)

对于 UI 金丝雀，setConfig 可包含以下布尔参数：

- continueOnStepFailure（布尔值）– 是否在步骤失败（指 executeStep 函数）后继续运行金丝雀脚本。如果任何步骤失败，金丝雀运行仍将被标记为失败。默认为 false。
- harFile（布尔值）– 是否创建 HAR 文件。默认为 True。
- screenshotOnStepStart（布尔值）– 是否在开始步骤之前捕获屏幕截图。
- screenshotOnStepSuccess（布尔值）– 是否在成功完成步骤后捕获屏幕截图。
- screenshotOnStepFailure（布尔值）– 是否在步骤失败后捕获屏幕截图。

启用或禁用屏幕截图的方法

disableStepScreenshots()

禁用所有屏幕截图选项（screenshotOnStepStart、screenshotOnStepSuccess 和 screenshotOnStepFailure）。

enableStepScreenshots()

启用所有屏幕截图选项（screenshotOnStepStart、screenshotOnStepSuccess 和 screenshotOnStepFailure）。预设情况下，这些方法均未启用。

getScreenshotOnStepFailure()

返回金丝雀是否在步骤失败后捕获屏幕截图。

getScreenshotOnStepStart()

返回金丝雀是否在开始步骤之前捕获屏幕截图。

getScreenshotOnStepSuccess()

返回金丝雀是否在成功完成步骤后捕获屏幕截图。

withScreenshotOnStepStart(screenshotOnStepStart)

接受一个布尔参数，该参数指示是否在开始步骤之前捕获屏幕截图。

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

接受一个布尔参数，该参数指示是否在成功完成步骤后捕获屏幕截图。

withScreenshotOnStepFailure(screenshotOnStepFailure)

接受一个布尔参数，该参数指示是否在步骤失败后捕获屏幕截图。

在 UI 金丝雀中的使用情况

首先，导入 Synthetics 依赖关系并获取配置。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

然后，通过使用以下选项之一调用 setConfig 方法来设置每个选项的配置。

```
// Set configuration values
    synConfig.setConfig({
        screenshotOnStepStart: true,
        screenshotOnStepSuccess: false,
```

```
        screenshotOnStepFailure: false
    });
```

Or

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnSt
```

若要禁用所有屏幕截图，请使用 disableStepScreenshots() 函数，如本示例所示。

```
synConfig.disableStepScreenshots();
```

您可以在代码中的任何位置启用和禁用屏幕截图。例如，若要仅禁用一个步骤的屏幕截图，请在运行该步骤之前禁用屏幕截图，然后在该步骤后启用它们。

用于 API 金丝雀的 setConfig(options)

对于 API 金丝雀，setConfig 可包含以下布尔参数：

- `continueOnHttpStepFailure`（布尔值）- 是否在 HTTP 步骤（指 executeHttpStep 函数）失败后继续运行金丝雀脚本。如果任何步骤失败，金丝雀运行仍将被标记为失败。默认为 `true`。

可视化监控

可视化监控将在金丝雀运行期间捕获的屏幕截图与在基准金丝雀运行期间捕获的屏幕截图进行比较。如果两个屏幕截图之间的差异超出阈值百分比，则金丝雀失败，您可以在金丝雀运行报告中看到以高亮颜色突出显示的差异区域。运行 syn-puppeteer-node-3.2 和更高版本的金丝雀支持可视化监控。运行 Python 和 Selenium 的金丝雀中目前不支持可视化监控。

若要启用可视化监控，请将以下代码行添加到金丝雀脚本中。有关更多详细信息，请参阅SyntheticsConfiguration 类。

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

将此行添加到脚本后，金丝雀首次成功运行时，它会使用在该运行期间捕获的屏幕截图作为比较基准。在金丝雀的该首次运行之后，您可以使用 CloudWatch 控制台编辑金丝雀以执行以下任一操作：

- 将金丝雀的下一次运行设置为新基准。
- 在当前基准屏幕截图上绘制边界，以指定在可视化比较过程中要忽略的屏幕截图区域。
- 删除屏幕截图，使其不用于可视化监控。

有关使用 CloudWatch 控制台编辑金丝雀的更多信息，请参阅 编辑或删除金丝雀脚本。

可视化监控的其他选项

syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)

设置可视化对比中屏幕截图差异的可接受百分比。

syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafa00")

设置在查看使用可视化监控的金丝雀运行报告时指明差异区域的突出显示颜色。

syntheticsConfiguration.withFailCanaryRunOnVisualVariance(failCanary)

设置当存在超过阈值的可视化差异时金丝雀是否失败。默认为金丝雀失败。

Synthetics Logger

SyntheticsLogger 将日志写入控制台和同一日志级别的本地日志文件。仅当日志级别与所调用的日志函数所需的日志记录级别相同或比其级别低时，此日志文件才会写入到这两个位置。

本地日志文件中的日志记录语句前面加上“DEBUG:”、“INFO:”等，以便与所调用的函数的日志级别相匹配。

如果您希望在与 Synthetics 金丝雀日志记录相同的日志级别运行 Synthetics 库，可以使用 SyntheticsLogger。

如果要创建上载到 S3 结果位置的日志文件，不需要使用 SyntheticsLogger。您可以在 `/tmp` 文件夹中创建不同的日志文件。在 `/tmp` 文件夹下创建的任何文件都会作为构件上传到 S3 中的结果位置。

要使用 Synthetics 库日志记录程序，请执行以下操作：

```
const log = require('SyntheticsLogger');
```

有用的函数定义：

log.debug(***message***, ***ex***);

参数：*message* 是要记录的消息。*ex* 是要记录的异常（如果有）。

例如：

```
log.debug("Starting step - login.");
```

log.error(***message***, ***ex***);

参数：*message* 是要记录的消息。*ex* 是要记录的异常（如果有）。

例如：

```
try {
  await login();
catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(***message***, ***ex***);

参数：*message* 是要记录的消息。*ex* 是要记录的异常（如果有）。

例如：

```
log.info("Successfully completed step - login.");
```

log.log(***message***, ***ex***);

这是 `log.info` 的一个别名。

参数：*message* 是要记录的消息。*ex* 是要记录的异常（如果有）。

例如：

```
 log.log("Successfully completed step - login.");
```

log.warn(***message***, ***ex***);

参数：*message* 是要记录的消息。*ex* 是要记录的异常（如果有）。

例如：

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper 类

`SyntheticsLogHelper` 类在 `syn-nodejs-puppeteer-3.2` 和更高版本的运行时中可用。其已在 CloudWatch Synthetics 库中初始化，并配置了 Synthetics 配置。您可以将它作为依赖项添加到脚本中。此类使您能够清理 URL、标头和错误消息，以编辑敏感信息。

> **Note**
>
> 根据 Synthetics 配置设置 `restrictedUrlParameters`，Synthetics 会对其记录的所有 URL 和错误消息进行清理，然后再将它们纳入日志、报告、HAR 文件和金丝雀运行错误中。仅当您在脚本中记录 URL 或错误时，才必须使用 `getSanitizedUrl` 或 `getSanitizedErrorMessage`。除了脚本引发的金丝雀错误之外，Synthetics 不会存储任何金丝雀构件。金丝雀运行构件存储在您的客户账户中。有关更多信息，请参阅 Synthetics 金丝雀的安全注意事项。

getSanitizedUrl(url, stepConfig = null)

此函数在 `syn-nodejs-puppeteer-3.2` 和更高版本中可用。它根据配置返回经过清理的 url 字符串。您可以通过设置 `restrictedUrlParameters` 属性选择编辑敏感 URL 参数（如密码和 access_token）。预设情况下，会编辑 URL 中的密码。如果需要，您可以通过将 `includeUrlPassword` 设置为 true 来启用 URL 密码。

如果传入的 URL 不是有效 URL，此函数会引发一个错误。

参数

- *`url`* 是一个字符串，也是要清理的 URL。
- *`StepConfig`*（可选）覆盖此函数的全局 Synthetics 配置。如果 `stepConfig` 未传入，则使用全局配置清理 URL。

示例

此示例使用以下示例 URL：`https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`。在此示例中，`access_token` 包含不应记录下来的敏感信息。请注意，Synthetics 服务不会存储任何金丝雀运行构件。日志、屏幕截图和报告等构件都存储在您客户账户的 Amazon S3 存储桶中。

第一步，设置 Synthetics 配置。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');
```

```
// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

接下来，清理并记录 URL

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?
access_token=12345&token_type=Bearer&expires_in=1200');
```

这一步会将以下内容记录在您的金丝雀日志中。

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

您可以通过传入包含 Synthetics 配置选项的可选参数来覆盖 URL 的 Synthetics 配置，如以下示例所示。

```
const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?
access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

上面的示例编辑了所有查询参数，并记录如下：

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

此函数在 `syn-nodejs-puppeteer-3.2` 和更高版本中可用。它根据 Synthetics 配置对所存在的任何 URL 进行清理，返回已清理的错误字符串。您可以选择在调用此函数时通过传递一个可选的 `stepConfig` 参数来覆盖全局 Synthetics 配置。

参数

- *error* 是要清理的错误。它可以是 Error 对象或字符串。
- *StepConfig*（可选）覆盖此函数的全局 Synthetics 配置。如果 stepConfig 未传入，则使用全局配置清理 URL。

示例

此示例使用以下内容：Failed to load url: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200

第一步，设置 Synthetics 配置。

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

接下来，清理并记录错误消息

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

try {
   // Your code which can throw an error containing url which your script logs
} catch (error) {
    const sanitizedErrorMessage = synthetics.getSanitizedErrorMessage(errorMessage);
    logger.info(sanitizedErrorMessage);
}
```

这一步会将以下内容记录在您的金丝雀日志中。

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

getSanitizedHeaders(headers, stepConfig=null)

此函数在 syn-nodejs-puppeteer-3.2 和更高版本中可用。它根据 syntheticsConfiguration 的 restrictedHeaders 属性返回经清理的标头。restrictedHeaders 属性中指定的标头是在日志、HAR 文件和报告中编辑的。

参数

- *headers* 是一个包含要清理的标题的对象。
- *StepConfig*（可选）覆盖此函数的全局 Synthetics 配置。如果 stepConfig 未传入，则使用全局配置来清理标头。

仅适用于 UI 金丝雀的 Node.js 库类和函数

以下用于 Node.js 的 CloudWatch Synthetics 库函数仅适用于 UI 金丝雀。

主题

- Synthetics 类
- BrokenLinkCheckerReport 类
- SyntheticsLink 类

Synthetics 类

以下函数位于 Synthetics 类中。

async addUserAgent(page, userAgentString);

此函数可将 *userAgentString* 附加到指定页面的 User-Agent 标头。

例如：

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

结果是页面的 User-Agent 标头被设置为 *browsers-user-agent-header-value*MyApp-1.0

async executeStep(stepName, functionToExecute, [stepConfig]);

执行提供的步骤，并使用开始/通过/失败日志记录、开始/通过/失败屏幕截图，以及通过/失败和持续时间指标对其进行包装。

> **Note**
>
> 如果您使用的是 syn-nodejs-2.1 或更高版本的运行时，您可以配置是否以及何时捕获屏幕截图。有关更多信息，请参阅 SyntheticsConfiguration 类。

executeStep 函数还执行以下操作：

- 记录步骤开始。
- 获取名为 <stepName>-starting 的屏幕截图。
- 启动计时器。
- 执行提供的函数。
- 如果函数正常返回，则计为通过。如果函数引发异常，则计为失败。
- 结束计时器。
- 记录步骤是通过还是失败
- 获取名为 <stepName>-succeeded 或 <stepName>-failed 的屏幕截图。
- 发出 stepName SuccessPercent 指标，100 表示通过，0 表示失败。
- 发出 stepName Duration 指标，指标的值基于步骤开始和结束时间。
- 最后，返回 functionToExecute 返回的内容或重新抛出 functionToExecute 抛出的内容。

如果金丝雀使用 syn-nodejs-2.0 或更高版本的运行时，则此函数还将步骤执行摘要添加到金丝雀的报告中。摘要包括有关每个步骤的详细信息，例如开始时间、结束时间、状态 (PASSED/FAILED)、故障原因（如失败）以及在每个步骤执行过程中捕获的屏幕截图。

例如：

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000)
 {
           await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});});
```

响应：

返回 functionToExecute 返回的内容。

syn-nodejs-2.2 的更新

从 syn-nodejs-2.2 开始，您可以选择传递步骤配置来在步骤级别覆盖 CloudWatch Synthetics 配置。有关可以传递给 executeStep 的选项列表，请参阅 SyntheticsConfiguration 类。

在以下示例中，用 true 覆盖 continueOnStepFailure 的默认 false 配置，并指定捕获屏幕截图的时间。

```
var stepConfig = {
    'continueOnStepFailure': true,
    'screenshotOnStepStart': false,
    'screenshotOnStepSuccess': true,
    'screenshotOnStepFailure': false
}

await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {
      await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});
 }, stepConfig);
```

getDefaultLaunchOptions();

getDefaultLaunchOptions() 函数返回 CloudWatch Synthetics 使用的浏览器启动选项。有关更多信息，请参阅 启动选项类型

```
// This function returns default launch options used by Synthetics.
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage();

返回当前打开的页面作为 Puppeteer 对象。有关更多信息，请参阅 Puppeteer API v1.14.0。

例如：

```
let page = synthetics.getPage();
```

响应：

在当前浏览器会话中打开的页面（Puppeteer 对象）。

getRequestResponseLogHelper();

> **Important**
>
> 在使用 `syn-nodejs-puppeteer-3.2` 或更高版本运行时的金丝雀中，此函数将与 `RequestResponseLogHelper` 类一起弃用。对此函数的任何使用都会导致在金丝雀日志中出现警告。将在未来的运行时版本中删除此函数。如果您正在使用此函数，请改为使用 RequestResponseLogHelper 类。

将此函数用作生成器模式，用于调整请求和响应日志记录标记。

例如：

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(fals
```

响应：

```
{RequestResponseLogHelper}
```

launch(options)

此函数的选项仅在 `syn-nodejs-2.1` 版本或更高版本的运行时中可用。

此函数仅用于 UI 金丝雀。它的作用是关闭现有浏览器并启动一个新的浏览器。

> **Note**
>
> 在开始运行脚本之前，CloudWatch Synthetics 始终启动浏览器。除非您想使用自定义选项启动新浏览器，否则不需要调用 launch()。

(options) 是在浏览器上设置的一组可配置的选项。有关更多信息，请参阅 启动选项类型。

如果您在不设置选项的情况下调用此函数，Synthetics 会以默认参数、`executablePath` 和 `defaultViewport` 启动浏览器。CloudWatch Synthetics 中的默认视区是 1920 x 1080。

您可以覆盖 CloudWatch Synthetics 使用的启动参数，并在启动浏览器时传递其他参数。例如，以下代码段以默认参数和默认可执行文件路径启动浏览器，但视区为 800 x 600。

```
await synthetics.launch({
```

```
        defaultViewport: {
            "deviceScaleFactor": 1,
            "width": 800,
            "height": 600
    }});
```

以下示例代码向 CloudWatch Synthetics 启动参数中添加了一个新的 `ignoreHTTPSErrors` 参数。

```
await synthetics.launch({
        ignoreHTTPSErrors: true
 });
```

您可以通过向 CloudWatch Synthetics 启动参数中的参数添加一个 `--disable-web-security` 标志来禁用 Web 安全：

```
// This function adds the --disable-web-security flag to the launch parameters
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
await synthetics.launch({
     args: launchArgs
  });
```

RequestResponseLogHelper 类

> **Important**
>
> 在使用 `syn-nodejs-puppeteer-3.2` 或更高版本运行时的金丝雀中，此类将被弃用。对此类的任何使用都会导致在金丝雀日志中出现警告。将在未来的运行时版本中删除此函数。如果您正在使用此函数，请改为使用 RequestResponseLogHelper 类。

处理精细配置，以及创建请求和响应负载的字符串表示。

```
class RequestResponseLogHelper {

    constructor () {
        this.request = {url: true, resourceType: false, method: false, headers: false,
 postData: false};
        this.response = {status: true, statusText: true, url: true, remoteAddress:
 false, headers: false};
    }
```

```
    withLogRequestUrl(logRequestUrl);

    withLogRequestResourceType(logRequestResourceType);

    withLogRequestMethod(logRequestMethod);

    withLogRequestHeaders(logRequestHeaders);

    withLogRequestPostData(logRequestPostData);


    withLogResponseStatus(logResponseStatus);

    withLogResponseStatusText(logResponseStatusText);

    withLogResponseUrl(logResponseUrl);

    withLogResponseRemoteAddress(logResponseRemoteAddress);

    withLogResponseHeaders(logResponseHeaders);
```

例如：

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()
.withLogRequestPostData(true)
.withLogRequestHeaders(true)
.withLogResponseHeaders(true));
```

响应：

```
{RequestResponseLogHelper}
```

setRequestResponseLogHelper();

> **Important**
>
> 在使用 `syn-nodejs-puppeteer-3.2` 或更高版本运行时的金丝雀中，此函数将与 `RequestResponseLogHelper` 类一起弃用。对此函数的任何使用都会导致在金丝雀日志中出现警告。将在未来的运行时版本中删除此函数。如果您正在使用此函数，请改为使用 RequestResponseLogHelper 类。

将此函数用作成生器模式，用于设置请求和响应日志记录标记。

例如：

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(tru
```

响应：

```
{RequestResponseLogHelper}
```

async takeScreenshot(name, suffix);

捕获当前页面的屏幕截图 (.PNG) 并命名和添加后缀（可选）。

例如：

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

在此示例中，捕获了名为 `01-navigateToUrl-loaded.png` 的屏幕截图并将其上载到金丝雀的 S3 存储桶。

您可以传递 `stepName` 作为第一个参数，捕获特定金丝雀步骤的屏幕截图。屏幕截图会链接到报告中的金丝雀步骤，以帮助您在调试时跟踪每个步骤。

CloudWatch Synthetics 金丝雀会在开始步骤之前（`executeStep` 函数）和步骤完成后自动捕获屏幕截图（除非您将金丝雀配置为禁用屏幕截图）。您可以通过将步骤名称传入 `takeScreenshot` 函数来捕获多个屏幕截图。

在以下示例中，捕获屏幕截图并将 `signupForm` 作为 `stepName` 的值。屏幕截图将被命名为 `02-signupForm-address`，并链接到金丝雀报告中名为 `signupForm` 的步骤。

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport 类

此类提供了添加 Synthetics 链接的方法。仅在使用 `syn-nodejs-2.0-beta` 或更高版本运行时的金丝雀上支持此类。

若要使用 `BrokenLinkCheckerReport`，脚本中需包括以下行：

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
```

```
const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

有用的函数定义：

addLink(*syntheticsLink*, isBroken)

syntheticsLink 是一个 SyntheticsLink 对象，表示链接。此函数根据状态代码添加链接。预设情况下，如果状态代码不可用或状态代码为 400 或以上，则其会将链接视为无效。您可以通过传入可选参数 isBrokenLink（带有值 true 或 false）来覆盖此默认行为。

此函数无返回值。

getLinks()

此函数返回一组 SyntheticsLink 对象，这些对象包含在无效链接检查器报告。

getTotalBrokenLinks()

此函数返回一个表示无效链接总数的数字。

getTotalLinksChecked()

此函数返回一个表示包含在报告中的链接总数的数字。

如何使用 BrokenLinkCheckerReport

以下金丝雀脚本代码段展示了导航到链接并将其添加到无效链接检查器报告的示例。

1. 导入 SyntheticsLink、BrokenLinkCheckerReport 和 Synthetics。

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
const SyntheticsLink = require('SyntheticsLink');

// Synthetics dependency
const synthetics = require('Synthetics');
```

2. 若要向报告添加链接，请创建 BrokenLinkCheckerReport 实例。

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. 导航到 URL 并将其添加到无效链接检查器报告中。

```
let url = "https://amazon.com";
```

```
let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();

// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
    const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout:
 30000});
} catch (ex) {
    // Add failure reason if navigation fails.
    link.withFailureReason(ex);
}

if (response) {
    // Capture screenshot of destination page
    let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

    // Add screenshot result to synthetics link
    link.addScreenshotResult(screenshotResult);

    // Add status code and status description to the link
    link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. 将报告添加到 Synthetics。这样，金丝雀每次运行都会在 S3 存储桶中创建一个名为 `BrokenLinkCheckerReport.json` 的 JSON 文件。您可以在控制台中查看金丝雀每次运行的链接报告以及屏幕截图、日志和 HAR 文件。

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink 类

此类提供了包装信息的方法。仅在使用 `syn-nodejs-2.0-beta` 或更高版本运行时的金丝雀上支持此类。

若要使用 `SyntheticsLink`，脚本中需包括以下行：

```
const SyntheticsLink = require('SyntheticsLink');

const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

此函数返回 syntheticsLink*Object*

有用的函数定义：

withUrl(***url***)

url 是一个 URL 字符串。此函数返回 syntheticsLink*Object*

withText(***text***)

text 是一个表示锚文本的字符串。此函数返回 syntheticsLink*Object*。它会添加与链接相对应的锚文本。

withParentUrl(***parentUrl***)

parentUrl 是一个表示父（源页面）URL 的字符串。此函数返回 syntheticsLink*Object*

withStatusCode(***statusCode***)

statusCode 是一个表示状态代码的字符串。此函数返回 syntheticsLink*Object*

withFailureReason(***failureReason***)

failureReason 是一个表示故障原因的字符串。此函数返回 syntheticsLink*Object*

addScreenshotResult(***screenshotResult***)

screenshotResult 是一个对象。它是 Synthetics 函数 takeScreenshot 返回的 ScreenshotResult 的实例。此对象包括以下属性：

- fileName – 表示 screenshotFileName 的字符串
- pageUrl（可选）
- error（可选）

仅适用于 API 金丝雀的 Node.js 库类和函数

以下用于 Node.js 的 CloudWatch Synthetics 库函数仅适用于 API 金丝雀。

主题

- executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

作为步骤执行提供的 HTTP 请求，并发布 `SuccessPercent`（通过/失败）和 `Duration` 指标。

executeHttpStep 在后台使用 HTTP 或 HTTPS 本机函数，具体取决于请求中指定的协议。

此函数还将步骤执行摘要添加到金丝雀的报告中。摘要中包括有关每个 HTTP 请求的详细信息，如下所示：

- 开始时间
- 结束时间
- 状态 (PASSED/FAILED)
- 故障原因（如失败）
- HTTP 调用详细信息，如请求/响应标头、请求/响应体、状态代码、状态消息和性能计时。

参数

stepName(***String***)

指定步骤的名称。此名称也用于发布此步骤的 CloudWatch 指标。

requestOptions(***Object or String***)

此参数的值可以是 URL、URL 字符串或对象。如果为对象，则必须是用以发出 HTTP 请求的一组可配置选项。其支持 Node.js 文档中 http.request(options[, callback]) 中的所有选项。

除了这些 Node.js 选项之外，requestOptions 支持附加参数 `body`。您可以使用 `body` 参数将数据作为请求体传递。

callback(***response***)

（可选）此函数是与 HTTP 响应一起调用的用户函数。响应的类型为 Class: http.IncomingMessage。

stepConfig(***object***)

（可选）使用此参数，可以用此步骤的不同配置覆盖全局 synthetics 配置。

executeHttpStep 的使用示例

以下一系列示例相互依存，以说明此选项的各种用途。

第一个示例展示配置请求参数。您可以将 URL 作为 requestOptions 传递：

```
let requestOptions = 'https://www.amazon.com';
```

或者，您也可以传递一组选项：

```
let requestOptions = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };
```

在下一个示例中，创建一个接受响应的回调函数。预设情况下，如果您未指定 callback，CloudWatch Synthetics 会验证状态是否介于 200 和 299 之间（含）。

```
// Handle validation for positive scenario
    const callback = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };
```

下一个示例展示为此步骤创建覆盖全局 CloudWatch Synthetics 配置的配置。在本示例中，步骤配置允许报告中包含请求标头、响应标头、请求体（发布数据）和响应体，并限制了“X-Amz-Security-Token”和“Authorization”标头值。预设情况下，出于安全原因，报告中不包括这些值。如果您选择将它们包含在报告中，则这些数据仅存储在 S3 存储桶中。

```
// By default headers, post data, and response body are not included in the report for
 security reasons.
// Change the configuration at global level or add as step configuration for individual
 steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
 values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

最后一个例子展示将请求传递给 executeHttpStep 并对步骤命名。

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
 stepConfig);
```

在这组示例中，CloudWatch Synthetics 将每个步骤的详细信息添加在报告中，并使用 stepName 生成每个步骤的指标。

您将看到 `Verify GET products API` 步骤的 `successPercent` 和 `duration` 指标。您可以通过监控 API 调用步骤的指标来监控 API 性能。

有关使用这些函数的完整脚本示例，请参阅 多步骤 API 金丝雀。

可用于使用 Selenium 的 Python 金丝雀脚本的库函数

本节列出了可用于 Python 金丝雀脚本的 Selenium 库函数。

主题

- 适用于所有金丝雀的 Python 和 Selenium 库类和函数
- 仅适用于 UI 金丝雀的 Python 和 Selenium 库类和函数

适用于所有金丝雀的 Python 和 Selenium 库类和函数

以下用于 Python 的 CloudWatch Synthetics 库函数适用于所有金丝雀。

主题

- SyntheticsConfiguration 类
- SyntheticsLogger 类

SyntheticsConfiguration 类

您可以使用 SyntheticsConfiguration 类来配置 Synthetics 库函数的行为。例如，您可以使用此类将 `executeStep()` 函数配置为不捕获屏幕截图。

您可以在全局级别设置 CloudWatch Synthetics 配置。

函数定义：

set_config(options)

```
from aws_synthetics.common import synthetics_configuration
```

`options` 是一个对象，它是一组适用于您的金丝雀的可配置选项。以下各节将介绍 *`options`* 中的可能字段。

- `screenshot_on_step_start`（布尔值）– 是否在开始步骤之前捕获屏幕截图。
- `screenshot_on_step_success`（布尔值）– 是否在成功完成步骤后捕获屏幕截图。
- `screenshot_on_step_failure`（布尔值）– 是否在步骤失败后捕获屏幕截图。

with_screenshot_on_step_start(screenshot_on_step_start)

接受一个布尔参数，该参数指示是否在开始步骤之前捕获屏幕截图。

with_screenshot_on_step_success(screenshot_on_step_success)

接受一个布尔参数，该参数指示是否在成功完成步骤后捕获屏幕截图。

with_screenshot_on_step_failure(screenshot_on_step_failure)

接受一个布尔参数，该参数指示是否在步骤失败后捕获屏幕截图。

get_screenshot_on_step_start()

返回是否在开始步骤之前捕获屏幕截图。

get_screenshot_on_step_success()

返回是否在成功完成步骤后捕获屏幕截图。

get_screenshot_on_step_failure()

返回是否在步骤失败后捕获屏幕截图。

disable_step_screenshots()

禁用所有屏幕截图选项（get_screenshot_on_step_start、get_screenshot_on_step_success 和 get_screenshot_on_step_failure）。

enable_step_screenshots()

启用所有屏幕截图选项（get_screenshot_on_step_start、get_screenshot_on_step_success 和 get_screenshot_on_step_failure)。预设情况下，这些方法均未启用。

CloudWatch 指标的相关 setConfig(options)

对于使用 `syn-python-selenium-1.1` 或更高版本的金丝雀，setConfig 的 (options) 可包含以下布尔参数，这些参数决定将由金丝雀发布的指标。其中每个选项的默认值均为 `true`。以 `aggregated` 开头的选项决定是否不带 `CanaryName` 维度发出指标。您可以使用该等指标查看所有金丝雀的聚合结果。其他选项决定是否带 `CanaryName` 维度发出指标。您可以使用该等指标查看每个金丝雀的结果。

有关金丝雀发出的 CloudWatch 指标的列表，请参阅 金丝雀发布的 CloudWatch 指标。

- `failed_canary_metric`（布尔值）– 是否发出此金丝雀的 `Failed` 指标（带 `CanaryName` 维度）。默认为 `true`。
- `failed_requests_metric`（布尔值）– 是否发出此金丝雀的 `Failed requests` 指标（带 `CanaryName` 维度）。默认为 `true`。
- `2xx_metric`（布尔值）– 是否发出此金丝雀的 `2xx` 指标（带 `CanaryName` 维度）。默认为 `true`。
- `4xx_metric`（布尔值）– 是否发出此金丝雀的 `4xx` 指标（带 `CanaryName` 维度）。默认为 `true`。
- `5xx_metric`（布尔值）– 是否发出此金丝雀的 `5xx` 指标（带 `CanaryName` 维度）。默认为 `true`。
- `step_duration_metric`（布尔值）– 是否发出此金丝雀的 `Step duration` 指标（带 `CanaryName StepName` 维度）。默认为 `true`。

- step_success_metric（布尔值）– 是否发出此金丝雀的 Step success 指标（带 CanaryName StepName 维度）。默认为 true。
- aggregated_failed_canary_metric（布尔值）– 是否发出此金丝雀的 Failed 指标（不带 CanaryName 维度）。默认为 true。
- aggregated_failed_requests_metric（布尔值）– 是否发出此金丝雀的 Failed Requests 指标（不带 CanaryName 维度）。默认为 true。
- aggregated_2xx_metric（布尔值）– 是否发出此金丝雀的 2xx 指标（不带 CanaryName 维度）。默认为 true。
- aggregated_4xx_metric（布尔值）– 是否发出此金丝雀的 4xx 指标（不带 CanaryName 维度）。默认为 true。
- aggregated_5xx_metric（布尔值）– 是否发出此金丝雀的 5xx 指标（不带 CanaryName 维度）。默认为 true。

with_2xx_metric(2xx_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 2xx 指标。

with_4xx_metric(4xx_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 4xx 指标。

with_5xx_metric(5xx_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 5xx 指标。

withAggregated2xxMetric(aggregated2xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 2xx 指标。

withAggregated4xxMetric(aggregated4xxMetric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 4xx 指标。

with_aggregated_5xx_metric(aggregated_5xx_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 5xx 指标。

with_aggregated_failed_canary_metric(aggregated_failed_canary_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 Failed 指标。

with_aggregated_failed_requests_metric(aggregated_failed_requests_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出不带维度的 Failed requests 指标。

with_failed_canary_metric(failed_canary_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Failed 指标。

with_failed_requests_metric(failed_requests_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Failed requests 指标。

with_step_duration_metric(step_duration_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 Duration 指标。

with_step_success_metric(step_success_metric)

接受一个布尔参数，该参数指定是否为此金丝雀发出带 CanaryName 维度的 StepSuccess 指标。

用于启用或禁用指标的方法

disable_aggregated_request_metrics()

禁用金丝雀发出以不带 CanaryName 维度形式发出的所有请求指标。

disable_request_metrics()

禁用所有请求指标，包括每个金丝雀的指标和跨所有金丝雀聚合的指标。

disable_step_metrics()

禁用所有步骤指标，包括步骤成功指标和步骤持续时间指标。

enable_aggregated_request_metrics()

启用金丝雀发出以不带 CanaryName 维度形式发出的所有请求指标。

enable_request_metrics()

启用所有请求指标，包括每个金丝雀的指标和跨所有金丝雀聚合的指标。

enable_step_metrics()

启用所有步骤指标，包括步骤成功指标和步骤持续时间指标。

在 UI 金丝雀中的使用情况

首先，导入 Synthetics 依赖关系并获取配置。然后，通过使用以下选项之一调用 setConfig 方法来设置每个选项的配置。

```
from aws_synthetics.common import synthetics_configuration

synthetics_configuration.set_config(
     {
        "screenshot_on_step_start": False,
        "screenshot_on_step_success": False,
        "screenshot_on_step_failure": True
     }
)

or
```

Or

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(F
```

若要禁用所有屏幕截图，请使用 disableStepScreenshots() 函数，如本示例所示。

```
synthetics_configuration.disable_step_screenshots()
```

您可以在代码中的任何位置启用和禁用屏幕截图。例如，若要仅禁用一个步骤的屏幕截图，请在运行该步骤之前禁用屏幕截图，然后在该步骤后启用它们。

用于 UI 金丝雀的 set_config(options)

从 `syn-python-selenium-1.1` 开始，对于 UI 金丝雀，`set_config` 可包含以下布尔参数:

- `continue_on_step_failure`（布尔值）– 是否在步骤失败（指 executeStep 函数）后继续运行金丝雀脚本。如果任何步骤失败，金丝雀运行仍将被标记为失败。默认为 `false`。

SyntheticsLogger 类

synthetics_logger 将日志写入控制台和同一日志级别的本地日志文件。仅当日志级别与所调用的日志函数所需的日志记录级别相同或比其级别低时，此日志文件才会写入到这两个位置。

本地日志文件中的日志记录语句前面加上“DEBUG:”、“INFO:”等，以便与所调用的函数的日志级别相匹配。

如果要创建上载到 Amazon S3 结果位置的日志文件，不需要使用 synthetics_logger。您可以在 /tmp 文件夹中创建不同的日志文件。在 /tmp 文件夹下创建的任何文件都会作为构件上载到 S3 存储桶中的结果位置。

若要使用 synthetics_logger：

```
from aws_synthetics.common import synthetics_logger
```

有用的函数定义：

获取日志级别：

```
log_level = synthetics_logger.get_level()
```

设置日志级别：

```
synthetics_logger.set_level()
```

使用指定级别记录消息。该级别可以是 DEBUG、INFO、WARN，或 ERROR，如以下语法示例所示：

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warn(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

有关调试参数的信息，请参阅标准 Python 文档中的 logging.debug

在这些日志记录函数中，message 为消息格式的字符串。args 为合并到使用字符串格式运算符的 msg 中的参数。

kwargs 中有三个关键字参数：

- exc_info – 如果未评估为 false，则会将异常信息添加到日志记录消息中。
- stack_info – 默认值为 false。如果为 true，则会将堆栈信息添加到日志记录消息中，包括实际的日志记录调用。
- extra – 第三个可选关键字参数，您可以使用该参数将用于填充 LogRecord（为具有用户定义属性的日志记录事件而创建）的 __dict__ 传入字典。

示例：

使用 DEBUG 级别记录消息：

```
synthetics_logger.debug('Starting step - login.')
```

使用 INFO 级别记录消息。logger.log 是 logger.info 的同义词：

```
synthetics_logger.info('Successfully completed step - login.')
```

或者

```
synthetics_logger.log('Successfully completed step - login.')
```

使用 WARN 级别记录消息：

```
synthetics_logger.warn('Warning encountered trying to publish %s', 'CloudWatch Metric')
```

使用 ERROR 级别记录消息：

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

记录异常：

```
synthetics_logger.exception(message, *args, **kwargs)
```

使用 ERROR 级别记录消息。异常信息将会添加到日志记录消息中。您应该只从异常处理程序调用此函数。

有关异常参数的信息，请参阅标准 Python 文档中的 logging.exception

message 为消息格式的字符串。args 为合并到使用字符串格式运算符的 msg 中的参数。

kwargs 中有三个关键字参数：

- exc_info – 如果未评估为 false，则会将异常信息添加到日志记录消息中。
- stack_info – 默认值为 false。如果为 true，则会将堆栈信息添加到日志记录消息中，包括实际的日志记录调用。
- extra – 第三个可选关键字参数，您可以使用该参数将用于填充 LogRecord（为具有用户定义属性的日志记录事件而创建）的 __dict__ 传入字典。

例如：

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch
 Metric')
```

仅适用于 UI 金丝雀的 Python 和 Selenium 库类和函数

以下用于 Python 的 CloudWatch Synthetics Selenium 库函数仅适用于 UI 金丝雀。

主题

- SyntheticsBrowser 类
- SyntheticsWebDriver 类

SyntheticsBrowser 类

当您通过调用 synthetics_webdriver.Chrome() 创建浏览器实例时，返回的浏览器实例类型为 SyntheticsBrowser。SyntheticsBrowser 类控制 ChromeDriver，并使金丝雀脚本能够驱动浏览器，从而允许 Selenium WebDriver 使用 Synthetics。

除了标准的 Selenium 方法之外，它还提供了以下方法。

set_viewport_size(width, height)

设置浏览器的视区。例如：

```
browser.set_viewport_size(1920, 1080)
```

save_screenshot(filename, suffix)

将屏幕截图保存到 /tmp 目录。屏幕截图将上载到 S3 存储桶中的金丝雀构件文件夹。

filename 是屏幕截图的文件名，后缀是用于命名屏幕截图的可选字符串。

例如：

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver 类

要使用此类，请在脚本中使用以下内容：

```
from aws_synthetics.selenium import synthetics_webdriver
```

add_execution_error(errorMessage, ex);

errorMessage 描述错误，ex 指遇到的异常

您可以使用 add_execution_error 来设置金丝雀的执行错误。它会在不中断脚本执行的情况下导致金丝雀失败。它也不会影响 successPercent 指标。

只有当错误对于指示金丝雀脚本成功或故障并不重要时，才应将错误作为执行错误进行跟踪。

以下是使用 add_execution_error 的示例。您将会监控端点的可用性，并在页面加载完成后捕获屏幕截图。由于捕获屏幕截图故障并不会决定端点的可用性，因此您可以捕获在截图时遇到的任何错误，并将它们添加为执行错误。可用性指标仍会指示端点已启动并正在运行，但金丝雀状态会被标记为失败。以下示例代码块将会捕获此类错误并将其添加为执行错误。

```
try:
    browser.save_screenshot("loaded.png")
except Exception as ex:
```

```
    self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

将 user_agent_str 的值附加到浏览器的用户代理标头。您必须在创建浏览器实例之前分配 user_agent_str。

例如：

```
synthetics_webdriver.add_user_agent('MyApp-1.0')
```

execute_step(step_name, function_to_execute)

处理一个函数。它还执行以下操作：

- 记录步骤开始。
- 获取名为 <stepName>-starting 的屏幕截图。
- 启动计时器。
- 执行提供的函数。
- 如果函数正常返回，则计为通过。如果函数引发异常，则计为失败。
- 结束计时器。
- 记录步骤是通过还是失败
- 获取名为 <stepName>-succeeded 或 <stepName>-failed 的屏幕截图。
- 发出 stepName SuccessPercent 指标，100 表示通过，0 表示失败。
- 发出 stepName Duration 指标，指标的值基于步骤开始和结束时间。
- 最后，返回 functionToExecute 返回的内容或重新抛出 functionToExecute 抛出的内容。

例如：

```
from selenium.webdriver.common.by import By

def custom_actions():
        #verify contains
        browser.find_element(By.XPATH, "//*[@id=\"id_1\"][contains(text(),'login')]")
        #click a button
        browser.find_element(By.XPATH, '//*[@id="submit"]/a').click()
```

```
    await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

Chrome()

启动 Chromium 浏览器的实例并返回创建的浏览器实例。

例如：

```
browser = synthetics_webdriver.Chrome()
browser.get("https://example.com/)
```

要以隐身模式启动浏览器，请遵循以下说明:

```
add_argument('—-incognito')
```

要添加代理设置，请遵循以下说明:

```
add_argument('--proxy-server=%s' % PROXY)
```

例如：

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("—-incognito")
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

使用 cron 安排金丝雀运行

当您安排金丝雀时，使用 cron 表达式可以让您灵活地安排计划。Cron 表达式包含五或六个按下表所列顺序排列的字段。这些字段采用空格分隔。语法根据您创建金丝雀使用的是 CloudWatch 控制台还是 AWS CLI 或 AWS SDK 而有所不同。若您使用控制台，则只能指定前五个字段。若使用 AWS CLI 或 AWS SDK，则可以指定所有六个字段，而且必须为 `Year` 字段指定 `*`。

字段	允许的值	允许的特殊字符
分钟	0-59	, - * /

字段	允许的值	允许的特殊字符
小时	0-23	, - * /
日期	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
星期几	1-7 或 SUN-SAT	, - * ? L #
年	*	

特殊字符

- ,（逗号）在字段的表达式中包含多个值。例如，在“Month（月份）”字段中，JAN、FEB 和 MAR 将包含 January、February 和 March。
- -（破折号）特殊字符用于指定范围。在“日”字段中，1-15 将包含指定月份的 1 - 15 日。
- *（星号）特殊字符包含该字段中的所有值。在“Hours（小时）”字段中，* 包括每个小时。您不能在同一个表达式的“Day-of-month（日期）”和“Day-of-week（星期几）”字段中同时使用 *。如果您在一个中使用它，则必须在另一个中使用 ?。
- /（正斜杠）用于指定增量。在“Minutes（分钟）”字段中，您可以输入 1/10 以指定从一个小时的第一分钟开始的每个第十分钟（例如，第 11 分钟、第 21 分钟和第 31 分钟，依此类推）。
- ?（问号）用于指定一个或另一个。在“日期”字段中，您可以输入 7，如果您不介意第 7 日是星期几，则可以在“星期几”字段中输入 ?。
- “日期”或“星期几”字段中的 L 通配符用于指定月或周的最后一天。
- “日期”字段中的 **W** 通配符用于指定工作日。在“日期”字段中，**3W** 用于指定最靠近当月的第三周的日。
- “星期几”字段中的 # 通配符用于指定一个月内所指定星期几的特定实例。例如，3#2 指该月的第二个星期二。3 指的是星期二，因为它是每周的第三天，2 是指该月内该类型的第二天。

限制

- 您无法在同一 Cron 表达式中为日期和星期几字段同时指定值。如果您在其中一个字段中指定值或 *（星号），则必须在另一个字段中使用 ?（问号）。
- 不支持产生的速率快于一分钟的 Cron 表达式。
- 您不能将金丝雀设置为等待一年以上才能运行，因此 `Year` 字段中只能指定为 *。

示例

创建金丝雀时，您可以参考以下示例 cron 字符串。以下示例展示了使用 AWS CLI 或 AWS SDK 创建或更新金丝雀的正确语法。如果您使用的是 CloudWatch 控制台，请省略每个示例末尾的 *。

Expression	含义
`0 10 * * ? *`	每天上午的 10:00 (UTC) 运行
`15 12 * * ? *`	每天在下午 12:15 (UTC) 运行
`0 18 ? * MON-FRI *`	每星期一到星期五的下午 6:00 (UTC) 运行
`0 8 1 * ? *`	每月第 1 天上午 8:00 (UTC) 运行
`0/10 * ? * MON-SAT *`	每周星期一到星期六每 10 分钟运行一次
`0/5 8-17 ? * MON-FRI *`	星期一到星期五的上午 8:00 和下午 5:55 (UTC) 之间，每 5 分钟运行一次

组

创建组可以将金丝雀相互关联，包括跨区域的金丝雀。使用组可以帮助您管理和自动化金丝雀，您还可以查看组中所有金丝雀的聚合运行结果和统计数据。

组是全局资源。创建组时，将在支持组的所有 AWS 区域中复制该组，您可以将其中任何一个区域的金丝雀添加到该组，并在这些区域中查看该组。虽然组 ARN 格式反映了其创建位置的区域名称，但组不受限于任何区域。这意味着您可以将来自多个区域的金丝雀放在同一组中，然后使用该组在单个视图中查看和管理所有这些金丝雀。

除默认禁用的区域外，所有区域都支持组。有关这些区域的更多信息，请参阅启用区域。

每组最多可以容纳 10 个金丝雀。您的账户中最多可以有 20 个组。任何一个金丝雀最多可以成为 10 个组的成员。

创建组

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。

3. 选择创建组。
4. 在 Group Name（组名称）下，输入组的名称。
5. 选择要与此组关联的金丝雀。要选择金丝雀，请在 Exact canary name（准确输入金丝雀名称）中键入其完整名称，然后选择 Search（搜索）。然后选中金丝雀名称旁边的复选框。如果不同区域有多个同名金丝雀，请务必选择所需的金丝雀。

   您可以重复此步骤，将最多 10 个金丝雀与该组关联。
6. （可选）在 Tags（标签）下，添加一个或多个键/值对作为此组的标签。标签可帮助您识别和组织 AWS 资源并跟踪 AWS 成本。有关更多信息，请参阅 标记 Amazon CloudWatch 资源。
7. 选择创建组。

在本地测试 Canary

本节介绍如何直接在 Microsoft Visual Studio 代码编辑器或 JetBrains IDE 代码编辑器中修改、测试和调试 CloudWatch Synthetics Canary。本地调试环境使用无服务器应用程序模型（SAM）容器来模拟 Lambda 函数，从而模拟 Synthetics Canary 的行为。

> **Note**
>
> 执行依赖于可视化监控的本地调试 Canary 是不切实际的。可视化监控依赖于在初始运行期间捕获基本屏幕截图，然后将这些屏幕截图与后续运行的屏幕截图进行比较。在本地开发环境中，运行不会被存储或跟踪，并且每次迭代都是独立的运行。由于缺少 Canary 运行历史记录，因此调试依赖可视化监控的 Canary 变得不切实际。

先决条件

1. 选择或创建 Amazon S3 存储桶，以用于存储本地 Canary 测试运行的构件，例如 HAR 文件和屏幕截图。这要求您预调配 IAM。如果您跳过设置 Amazon S3 存储桶，您仍然可以在本地测试您的 Canary，但是您将看到一条关于丢失存储桶的错误消息，并且您将无法访问 Canary 构件。

   如果您使用 Amazon S3 存储桶，我们建议您将存储桶生命周期设置为在几天后删除对象，以节省成本。有关更多信息，请参阅管理存储生命周期。
2. 为您的 AWS 账户设置默认 AWS 配置文件。有关更多信息，请参阅配置和凭证文件设置。
3. 将调试环境的默认 AWS 区域设置为您的首选区域，例如 `us-west-2`。
4. 安装 AWS SAM CLI。有关更多信息，请参阅安装 AWS SAM CLI。

5. 安装 Visual Studio Code Editor 或 JetBrains IDE。有关更多信息，请参阅 Visual Studio Code 或 JetBrains IDE。
6. 安装 Docker 以使用 AWS SAM CLI。请务必启动 docker 进程守护程序。有关更多信息，请参阅安装 Docker 以用于 AWS SAM CLI。

   或者，您可以安装其他容器管理软件，例如 Rancher，只要它使用 Docker 运行时即可。
7. 为您的首选编辑器安装 AWS 工具包扩展。有关更多信息，请参阅安装 AWS Toolkit for Visual Studio Code 或安装 AWS Toolkit for JetBrains。

主题

- 设置测试和调试环境
- 使用 Visual Studio Code IDE
- 使用 JetBrains IDE
- 使用 SAM CLI 在本地运行 Canary
- 将您的本地测试环境集成到现有的 Canary 包中
- 更改 CloudWatch Synthetics 运行时
- 常见错误

设置测试和调试环境

首先，通过输入以下命令克隆 AWS 提供的 Github 存储库。该存储库包含 Node.js Canary 和 Python Canary 的代码示例。

```
git clone https://github.com/aws-samples/synthetics-canary-local-debugging-sample.git
```

然后根据您的 Canary 的语言执行以下操作之一。

对于 Node.js Canary

1. 通过输入以下命令转到 Node.js Canary 源目录。

   ```
   cd synthetics-canary-local-debugging-sample/nodejs-canary/src
   ```
2. 运行以下命令，以安装 Canary 依赖项。

   ```
   npm install
   ```

对于 Python Canary

1. 通过输入以下命令转到 Python Canary 源目录。

```
cd synthetics-canary-local-debugging-sample/python-canary/src
```

2. 运行以下命令，以安装 Canary 依赖项。

```
pip3 install -r requirements.txt -t .
```

使用 Visual Studio Code IDE

Visual Studio 启动配置文件位于 `.vscode/launch.json`。它包含允许 Visual Studio 代码发现模板文件的配置。它定义了一个 Lambda 有效负载，其中包含成功调用 Canary 所需的参数。以下是 Node.js Canary 的启动配置：

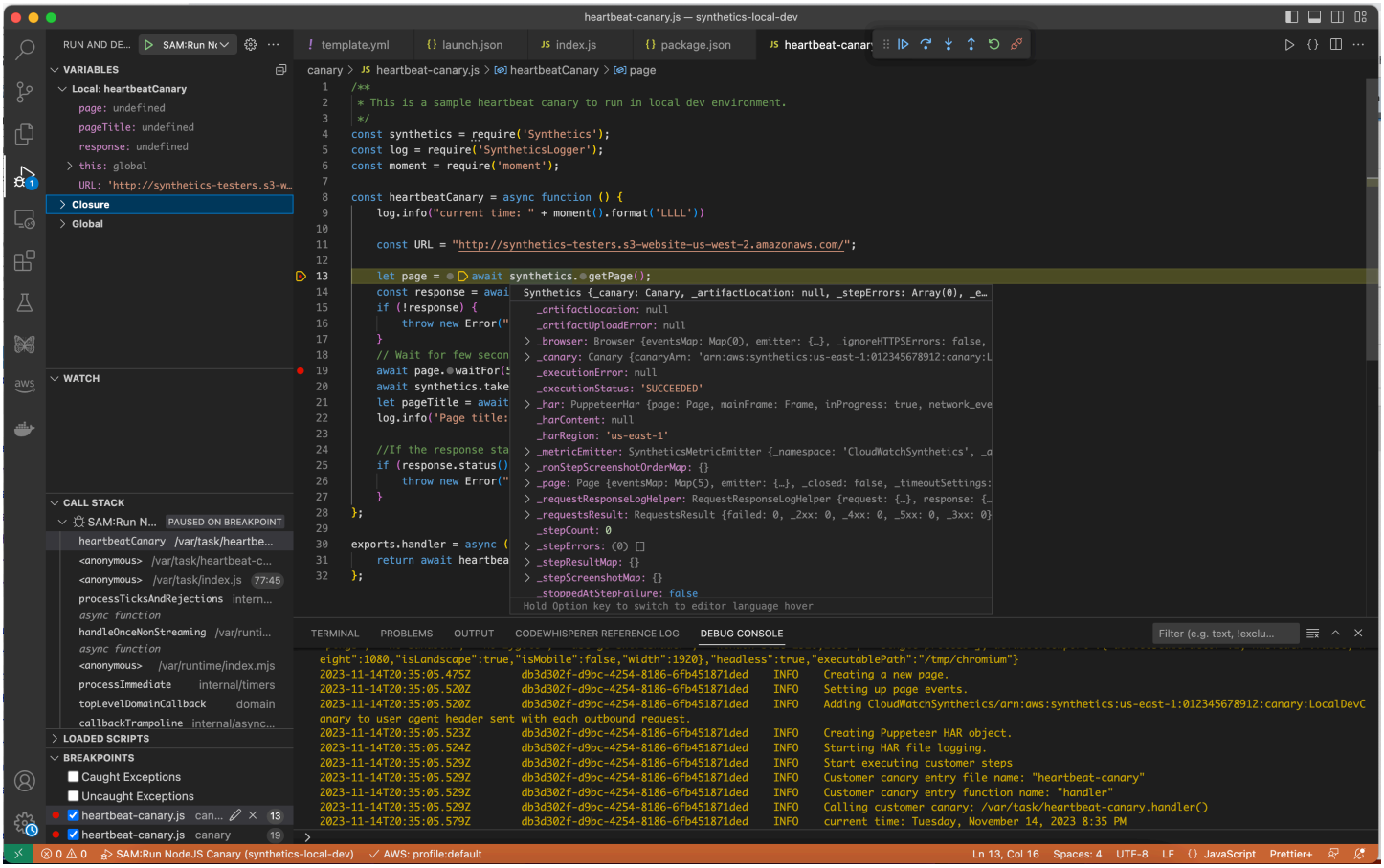
```
{
            ...
            ...
            "lambda": {
                "payload": {
                    "json": {
                        // Canary name. Provide any name you like.
                        "canaryName": "LocalSyntheticsCanary",
                        // Canary artifact location
                        "artifactS3Location": {
                            "s3Bucket": "cw-syn-results-123456789012-us-west-2",
                            "s3Key": "local-run-artifacts",
                        },
                        // Your canary handler name
                        "customerCanaryHandlerName": "heartbeat-canary.handler"
                    }
                },
                // Environment variables to pass to the canary code
                "environmentVariables": {}
            }
        }
    ]
}
```

您也可以选择在有效负载 JSON 中提供以下字段：

- s3EncryptionMode 有效值：SSE_S3 | SSE_KMS
- s3KmsKeyArn 有效值：*KMS ## ARN*
- activeTracing 有效值：true | false
- canaryRunId 有效值：*UUID* 如果启用了主动跟踪，则需要此参数。

要在 Visual Studio 中调试 Canary，请在要暂停执行的 Canary 代码中添加断点。要添加断点，请选择编辑器边距，然后在编辑器中进入运行和调试模式。点击播放按钮运行 Canary。当 Canary 运行时，将在调试控制台中跟踪日志，从而为您提供有关 Canary 行为的实时见解。如果您添加了断点，则 Canary 执行将在每个断点处暂停，从而使您可以逐步浏览代码并检查变量值、实例方法、对象属性和函数调用堆栈。

除了存储在 Amazon S3 存储桶中的构件和每次本地运行生成的 CloudWatch 指标外，在本地运行和调试 Canary 不会产生任何费用。



使用 JetBrains IDE

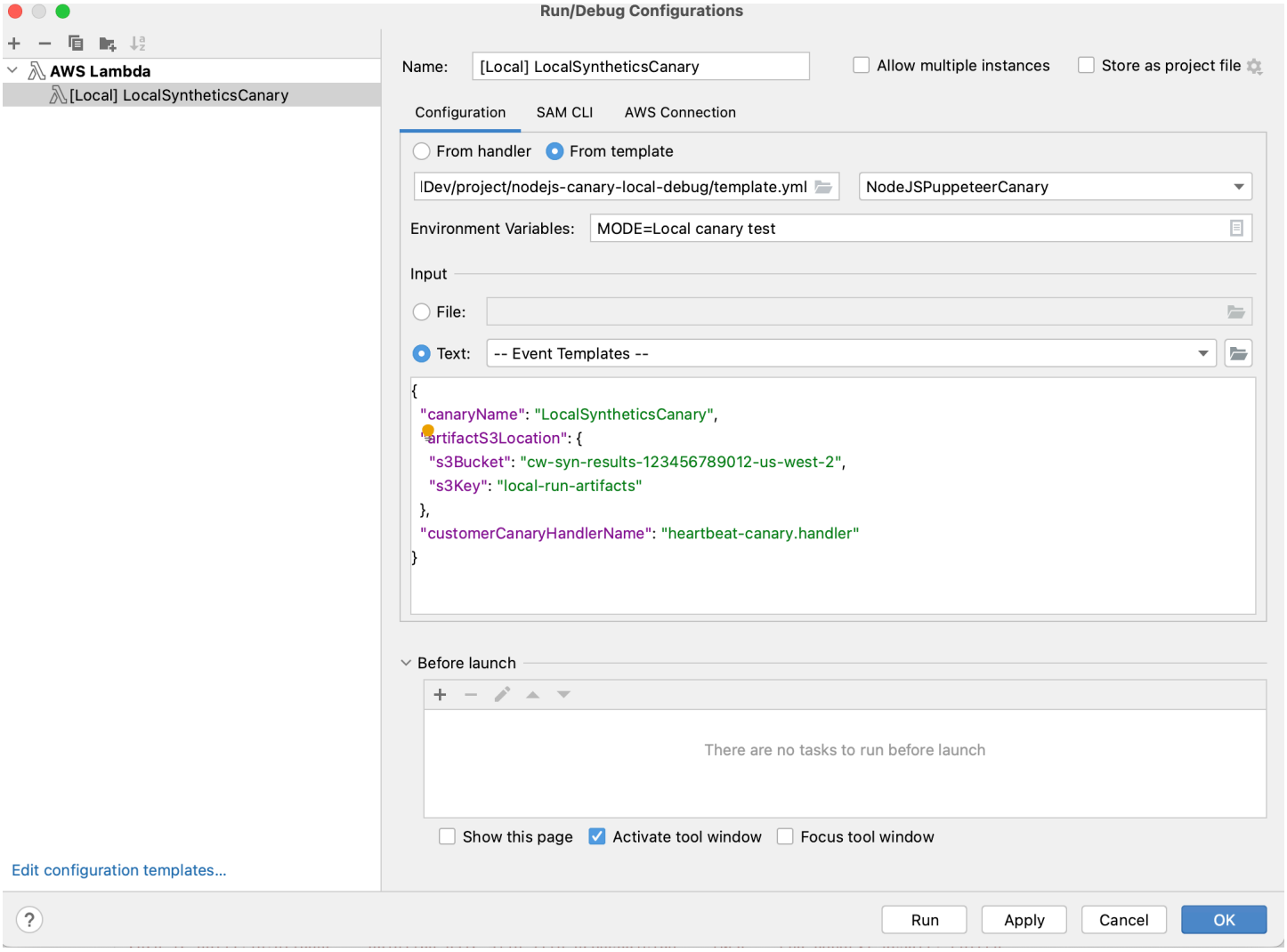
安装 AWS Toolkit for JetBrains 扩展程序后，如果您正在调试 Node.js Canary，请确保启用 Node.js 插件和 JavaScript 调试器运行。然后，按照以下步骤执行操作。

使用 JetBrains IDE 调试 Canary

1. 在 JetBrains IDE 的左侧导航窗格中，选择 Lambda，然后选择本地配置模板。
2. 为运行配置输入名称，例如 **`LocalSyntheticsCanary`**
3. 选择来自模板，在模板字段中选择文件浏览器，然后从 nodejs 目录或 python 目录中在项目中选择 template.yml 文件。
4. 在输入部分，输入 Canary 的有效负载，如以下屏幕所示。

```
{
 "canaryName": "LocalSyntheticsCanary",
 "artifactS3Location": {
     "s3Bucket": "cw-syn-results-123456789012-us-west-2",
     "s3Key": "local-run-artifacts"
 },
 "customerCanaryHandlerName": "heartbeat-canary.handler"
}
```

您也可以在有效负载 JSON 中设置其他环境变量，如 使用 Visual Studio Code IDE 中所列。



使用 SAM CLI 在本地运行 Canary

使用以下过程之一，使用无服务器应用程序模型（SAM）CLI 在本地运行 Canary。请务必在 `event.json` 中为 `s3Bucket` 指定您自己的 Amazon S3 存储桶名称

要使用 SAM CLI 运行 Node.js Canary

1. 通过输入以下命令转到源目录。

```
cd synthetics-canary-local-debugging-sample/nodejs-canary
```

2. 输入以下命令。

```
sam build
sam local invoke -e ../event.json
```

要使用 SAM CLI 运行 Python Canary

1. 通过输入以下命令转到源目录。

```
cd synthetics-canary-local-debugging-sample/python-canary
```

2. 输入以下命令。

```
sam build
sam local invoke -e ../event.json
```

将您的本地测试环境集成到现有的 Canary 包中

您可以通过复制三个文件将本地 Canary 调试集成到现有的 Canary 包中：

- 将 `template.yml` 文件复制到您的 Canary 包根目录中。请务必修改 `CodeUri` 的路径以指向您的 Canary 代码所在的目录。
- 如果您使用的是 Node.js Canary，请将 `cw-synthetics.js` 文件复制到您的 Canary 源目录。如果您使用的是 Python Canary，请将 `cw-synthetics.py` 复制到您的 Canary 源目录。
- 复制启动配置文件。`vscode/launch.json` 进入包根目录。确保将其放在 `.vscode` 目录中；如果目录尚不存在，请创建目录。

更改 CloudWatch Synthetics 运行时

在调试的过程中，您可能需要尝试使用不同的 CloudWatch Synthetics 运行时来运行 Canary，而不是最新的运行时。为此，请从下表之一中找到要使用的运行时。务必为正确的区域选择运行时。然后，将该运行时的 ARN 粘贴到 `template.yml` 文件中的相应位置，然后运行 Canary。

Node.js 运行时

syn-nodejs-puppeteer-7.0 的 ARN

下表列出了要在各个 AWS 区域中用于 `syn-nodejs-puppeteer-7.0` 版本 CloudWatch Synthetics 运行时的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:44
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:46
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:44
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:47
非洲（开普敦）	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:44
亚太地区（香港）	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:45
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:20
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:26
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:18
亚太地区（孟买）	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:44
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:30
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:46

区域	ARN
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:49
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:44
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:44
加拿大（中部）	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:44
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:76
中国（北京）	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:45
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:46
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:44
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:46
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:44
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:45
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:44
欧洲（西班牙）	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:20

区域	ARN
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:44
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:19
以色列（特拉维夫）	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:17
中东（巴林）	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:44
中东（阿联酋）	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:19
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:45
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:41
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:42

syn-nodejs-puppeteer-6.2 的 ARN

下表列出了要在各个 AWS 区域中用于 syn-nodejs-puppeteer-6.2 版本 CloudWatch Synthetics 运行时的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:41
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:43

区域	ARN
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:41
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:44
非洲（开普敦）	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:41
亚太地区（香港）	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:42
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
亚太地区（孟买）	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:41
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:42
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:46
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:41
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:41

区域	ARN
加拿大（中部）	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:41
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
中国（北京）	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:42
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:43
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:41
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:43
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:41
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:42
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:41
欧洲（西班牙）	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:41
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
以色列（特拉维夫）	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14

区域	ARN
中东（巴林）	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:41
中东（阿联酋）	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:42
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:39
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:39

syn-nodejs-puppeteer-5.2 的 ARN

下表列出了要在各个 AWS 区域中用于 syn-nodejs-puppeteer-5.2 版本 CloudWatch Synthetics 运行时的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:42
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:44
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:42
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:45
非洲（开普敦）	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:42

区域	ARN
亚太地区（香港）	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:43
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:18
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:24
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:16
亚太地区（孟买）	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:42
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:28
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:44
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:47
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:42
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:42
加拿大（中部）	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:42
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:74
中国（北京）	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:43

区域	ARN
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:44
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:42
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:44
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:42
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:43
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:42
欧洲（西班牙）	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:18
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:42
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:17
以色列（特拉维夫）	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:15
中东（巴林）	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:42
中东（阿联酋）	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:17
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:43

区域	ARN
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:40
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:40

Python 运行时

syn-python-selenium-3.0 的 ARN

下表列出了要在各个 AWS 区域中用于 `syn-python-selenium-3.0` 版本 CloudWatch Synthetics 运行时的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	aarn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:32
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:34
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:32
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:34
非洲（开普敦）	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:32
亚太地区（香港）	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:32
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:20

区域	ARN
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:26
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:18
亚太地区（孟买）	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:32
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:30
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:34
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:37
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:32
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:32
加拿大（中部）	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:32
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:76
中国（北京）	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:32
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:32
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:32

区域	ARN
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:34
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:32
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:33
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:32
欧洲（西班牙）	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:20
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:32
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:19
以色列（特拉维夫）	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:17
中东（巴林）	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:32
中东（阿联酋）	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:19
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:33
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:30
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:31

syn-python-selenium-2.1 的 ARN

下表列出了要在各个 AWS 区域中用于 syn-python-selenium-2.1 版本 CloudWatch Synthetics 运行时的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:29
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:31
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:29
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:31
非洲（开普敦）	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:29
亚太地区（香港）	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:29
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
亚太地区（孟买）	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:29
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27

区域	ARN
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:30
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:34
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:29
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:29
加拿大（中部）	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:29
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
中国（北京）	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:29
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:29
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:29
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:31
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:29
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:30
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:29

区域	ARN
欧洲（西班牙）	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:29
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
以色列（特拉维夫）	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
中东（巴林）	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:29
中东（阿联酋）	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:30
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:29
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:29

常见错误

错误：在本地运行 AWS SAM 项目需要 Docker。您是否已安装并正在运行它？

请务必在计算机上启动 Docker。

SAM 本地调用失败：调用 GetLayerVersion 操作时发生错误（ExpiredTokenException）：请求中包含的安全令牌已过期

确保已设置 AWS 默认配置文件。

更常见的错误

有关 SAM 常见错误的更多信息，请参阅 AWS SAM CLI 故障排除。

排查失败金丝雀的问题

如果您的金丝雀失败，请检查以下内容以排查问题。

一般故障排除

- 使用金丝雀详细信息页面查找更多信息。在 CloudWatch 控制台中，选择导航窗格中的 Canaries（金丝雀），然后选择金丝雀的名称以打开金丝雀详细信息页。在 Availability（可用性）选项卡上，选中 SuccessPercent 指标以查看问题是经常性问题还是间歇性问题。

  仍在 Availability（可用性）选项卡上时，选择失败的数据点以查看该失败运行的屏幕截图、日志和步骤报告（如可用）。

  如果步骤报告可用（因为步骤是脚本的一部分），请检查以查明失败的步骤，并查看相关屏幕截图以查看客户看到的问题。

  您还可以检查 HAR 文件以了解是否有一个或多个请求失败。您可以使用日志对失败的请求和错误进行深入分析。最后，您可以将这些构件与成功金丝雀运行中的构件进行比较，以查明问题。

  预设情况下，CloudWatch Synthetics 会捕获 UI 金丝雀中每个步骤的屏幕截图。但您的脚本可能会配置为禁用屏幕截图。在调试期间，您可能希望再次启用屏幕截图。同样，对于 API 金丝雀，您可能希望在调试过程中查看 HTTP 请求和响应标头和响应体。有关如何将此数据包括在报告中，请参阅 executeHttpStep(stepName, requestOptions, [callback], [stepConfig])。
- 如果您最近对应用程序做出了部署，请执行回滚操作，然后稍后进行调试。
- 手动连接到端点，看看是否会重现相同问题。

主题

- 金丝雀在 Lambda 环境更新后发生故障
- 我的金丝雀被 AWS WAF 阻止了
- 等待元素出现
- 节点不可见或不是用于 page.click() 的 HTMLElement
- 无法将构件上载到 S3，“异常：无法获取 S3 存储桶位置：访问被拒绝”
- “错误：协议错误 (Runtime.callFunctionOn)：目标已关闭。”

- “金丝雀失败。错误：无数据点 – 金丝雀显示超时”错误
- 尝试访问内部端点
- 金丝雀运行时版本升级和降级问题
- 跨域请求共享 (CORS) 问题
- 金丝雀争用情况问题
- 排查 VPC 上金丝雀的问题

金丝雀在 Lambda 环境更新后发生故障

CloudWatch Synthetics 金丝雀在您的账户中作为 Lambda 函数实施。这些 Lambda 函数需要定期更新 Lambda 运行时系统，其中包括安全更新、错误修复和其他改进。Lambda 致力于提供与现有函数向后兼容的运行时系统更新。但是，与软件修补一样，在极少数情况下，运行时更新会对现有函数产生负面影响。如果您认为自己的金丝雀受到 Lambda 运行时系统更新的影响，则可以使用 Lambda 运行时系统管理手动模式（在支持的区域中）来临时回滚 Lambda 运行时系统版本。这样，可以使金丝雀函数保持正常运行并最大限度地减少中断，从而在返回到最新的运行时系统版本之前有时间解决不兼容问题。

如果您的金丝雀在 Lambda 运行时系统更新后出现故障，最好的解决方案是升级到最新的 Synthetics 运行时系统之一。有关最新运行时系统的更多信息，请参阅 Synthetics 运行时版本。

作为替代解决方案，在提供 Lambda 运行时系统管理控件的区域中，您可以使用手动模式进行运行时系统管理控制，将金丝雀恢复到较旧的 Lambda 托管运行时系统。您可以按照以下各部分中的以下步骤使用 AWS CLI 或通过 Lambda 控制台设置手动模式。

> **Warning**
>
> 当您将运行时系统设置更改为手动模式时，您的 Lambda 函数在恢复到自动模式之前不会收到自动安全更新。在此期间，您的 Lambda 函数可能容易受到安全漏洞的影响。

先决条件

- 安装 jq
- 安装最新版本的 AWS CLI。有关更多信息，请参阅 AWS CLI 安装和更新说明。

步骤 1：获取 Lambda 函数 ARN

运行以下命令，以从响应中检索 EngineArn 字段。此 EngineArn 是与金丝雀关联的 Lambda 函数的 ARN。您将在以下步骤中使用此 ARN。

```
aws synthetics get-canary --name my-canary | jq '.Canary.EngineArn'
```

EngingArn 的示例输出：

```
"arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-
afb1-EXAMPLE991:8"
```

步骤 2：获取最后一个良好的 Lambda 运行时系统版本 ARN

为了帮助了解您的金丝雀是否受到 Lambda 运行时系统更新的影响，请检查日志中 Lambda 运行时系统版本 ARN 更改的日期和时间是否与您看到金丝雀受到影响的日期和时间一致。如果两者不匹配，则可能不是 Lambda 运行时系统更新导致您的问题。

如果您的金丝雀受到 Lambda 运行时系统更新的影响，则必须识别之前正在使用的有效 Lambda 运行时系统版本的 ARN。按照识别运行时系统版本更改中的说明查找先前的运行时系统的 ARN。记录运行时系统版本 ARN，然后继续执行步骤 3，以设置运行时系统管理配置。

如果您的金丝雀尚未受到 Lambda 环境更新的影响，则可以查找您当前正在使用的 Lambda 运行时系统版本的 ARN。运行以下命令，以从响应中检索 Lambda 函数的 RuntimeVersionArn。

```
aws lambda get-function-configuration \
--function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991:8" | jq '.RuntimeVersionConfig.RuntimeVersionArn'
```

RuntimeVersionArn 的示例输出：

```
"arn:aws:lambda:us-
west-2::runtime:EXAMPLE647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

步骤 3：更新 Lambda 运行时系统管理配置

您可以使用 AWS CLI 或 Lambda 控制台来更新运行时系统管理配置。

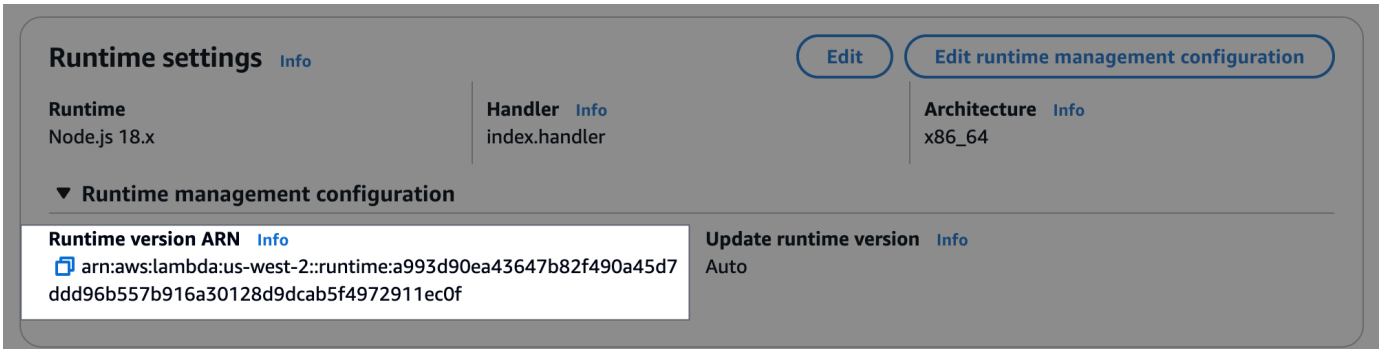
要设置 Lambda 运行时系统管理配置手动模式，请使用 AWS CLI

输入以下命令，以将 Lambda 函数的运行时系统管理更改为手动模式。请务必使用您在步骤 1 中找到的值，将 *function-name* 和 *qualifier* 分别替换为 Lambda 函数 ARN 和 Lambda 函数版本号。另外，将 *runtime-version-arn* 替换为您在步骤 2 中找到的版本 ARN。

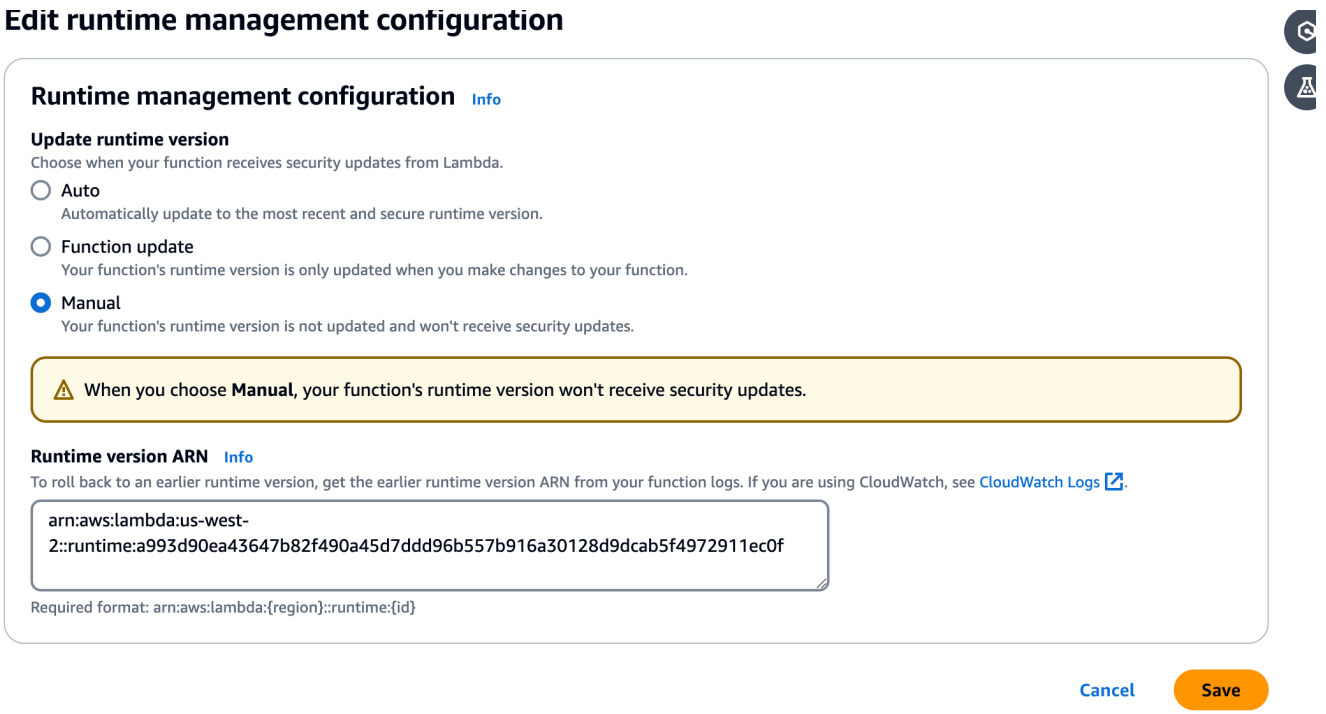
```
aws lambda put-runtime-management-config \
    --function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-afb1-EXAMPLE991" \
    --qualifier 8 \
    --update-runtime-on "Manual" \
    --runtime-version-arn "arn:aws:lambda:us-west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

要使用 Lambda 控制台将金丝雀更改为手动模式

1. 通过 https://console.aws.amazon.com/lambda/ 打开 AWS Lambda 控制台。
2. 选择版本选项卡，选择与您的 ARN 对应的版本号链接，然后选择代码选项卡。
3. 向下滚动到运行时系统设置，展开运行时系统管理配置，然后复制运行时系统版本 ARN。



4. 选择编辑运行时系统管理配置，选择手动，将之前复制的运行时系统版本 ARN 粘贴到运行时系统版本 ARN 字段中。然后选择保存。



我的金丝雀被 AWS WAF 阻止了

为了防止 AWS WAF 阻止您的金丝雀，请设置一个允许使用字符串 `CloudWatchSynthetics` 的 AWS WAF 字符串匹配条件。有关更多信息，请参阅 AWS WAF 文档中的使用字符串匹配条件。

等待元素出现

分析日志和屏幕截图后，如果您发现脚本正在等待某个元素出现在屏幕上并超时，请检查相关的屏幕截图以查看该元素是否出现在页面上。验证 `xpath` 以确保该元素的准确性。

对于与 Puppeteer 相关的问题，请参阅 Puppeteer 的 GitHub 页面或 Internet 论坛。

节点不可见或不是用于 page.click() 的 HTMLElement

如果节点不可见或不是用于 `page.click()` 的 `HTMLElement`，请首先验证您正在用于单击元素的 `xpath`。另外，如果您的元素位于屏幕底部，请调整视区。预设情况下，CloudWatch Synthetics 采用的视区为 1920 * 1080。您可以在启动浏览器时或使用 Puppeteer 函数 `page.setViewport` 另行设置视区。

无法将构件上载到 S3，“异常：无法获取 S3 存储桶位置：访问被拒绝”

如果您的金丝雀因 Amazon S3 错误而失败，CloudWatch Synthetics 无法上传屏幕截图、日志或因权限问题为金丝雀创建的报告。请检查以下事项：

- 检查金丝雀的 IAM 角色是否具有 `s3:ListAllMyBuckets` 权限、对正确的 Amazon S3 存储桶的 `s3:GetBucketLocation` 权限，以及对金丝雀存储其构件的存储桶的 `s3:PutObject` 权限。如果金丝雀执行可视化监控，该角色还需要存储桶的 `s3:GetObject` 权限。如果 Canary 部署在具有 VPC 端点的 VPC 中，则 Amazon VPC S3 网关端点策略也需要相同权限。
- 如果金丝雀使用 AWS KMS 客户托管式密钥而不是标准 AWS 托管式密钥（默认）来加密，金丝雀的 IAM 角色可能没有使用该密钥加密或解密的权限。有关更多信息，请参阅 加密金丝雀构件。
- 您的存储桶策略可能不允许金丝雀使用的加密机制。例如，如果您的存储桶策略要求使用特定的加密机制或 KMS 密钥，则必须为金丝雀选择相同的加密模式。

如果金丝雀执行可视化监控，请参阅 使用可视化监控时更新构件位置和加密 了解更多信息。

“错误：协议错误 (Runtime.callFunctionOn)：目标已关闭。”

如果在页面或浏览器关闭后还有一些网络请求，则会出现此错误。您可能忘记了等待异步操作。执行脚本后，CloudWatch Synthetics 会关闭浏览器。关闭浏览器后执行任何异步操作可能会导致 `target closed error` 错误。

“金丝雀失败。错误：无数据点 – 金丝雀显示超时”错误

此错误指金丝雀运行超过了超时时间。金丝雀执行在 CloudWatch Synthetics 发布成功率 CloudWatch 指标或更新构件（如 HAR 文件、日志和屏幕截图）前已经停止。如果超时时间太低，您可以将其提高。

预设情况下，金丝雀的超时值等于其频率。您可以手动将超时值调整为小于或等于金丝雀频率。如果金丝雀频率较低，则必须增加频率以提高超时时间。在使用 CloudWatch Synthetics 控制台创建或更新金丝雀时，您可以在 Schedule（计划）下调整频率和超时值。

请确保金丝雀超时值不要短于 15 秒钟，以预留 Lambda 冷启动以及启动金丝雀工具的时间。

发生此错误时，无法在 CloudWatch Synthetics 控制台中查看金丝雀构件。您可以使用 CloudWatch Logs 查看金丝雀的日志。

使用 CloudWatch Logs 查看金丝雀的日志

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，选择 Log groups（日志组）。
3. 在筛选器框中键入金丝雀名称来查找日志组。金丝雀的日志组名称为 /aws/lambda/cwsyn-***canaryName***-randomId。

尝试访问内部端点

如果希望您的金丝雀访问内部网络上的端点，我们建议您将 CloudWatch Synthetics 设置为使用 VPC。有关更多信息，请参阅 在 VPC 上运行金丝雀。

金丝雀运行时版本升级和降级问题

如果您最近将金丝雀的运行时版本从 `syn-1.0` 升级到更高版本，则可能是跨域请求共享 (CORS) 问题。有关更多信息，请参阅 跨域请求共享 (CORS) 问题。

如果您最近将金丝雀运行时版本降级到较旧版本，请检查以确保您正在使用的 CloudWatch Synthetics 函数在您降级到的旧运行时版本中可用。例如，`executeHttpStep` 函数可用于 `syn-nodejs-2.2` 或更高版本的运行时。要检查函数的可用性，请参阅 编写金丝雀脚本。

> **Note**
>
> 当您计划升级或降级金丝雀的运行时系统版本时，我们建议您首先克隆金丝雀并更新克隆金丝雀中的运行时系统版本。在验证了使用新运行时版本的克隆金丝雀正常运行后，您可以更新原始金丝雀的运行时版本并删除克隆金丝雀。

跨域请求共享 (CORS) 问题

在 UI 金丝雀中，如果某些网络请求失败并显示 `403` 或 `net::ERR_FAILED` 错误，请检查金丝雀是否启用了活动跟踪，并使用 Puppeteer 函数 `page.setExtraHTTPHeaders` 添加标头。若是如此，则网络请求失败可能是由跨域请求共享 (CORS) 限制所致。您可以通过禁用活动跟踪或删除额外的 HTTP 标头来确认是否属于这种情况。

为什么会发生这种情况？

在使用活动跟踪时，会向所有传出请求添加额外的标头以跟踪调用。通过使用 Puppeteer 的 `page.setExtraHTTPHeaders` 来添加跟踪标头或添加额外标头的方式来修改请求标头，会导致对 XMLHttpRequest (XHR) 请求进行 CORS 检查。

如果您不想禁用活动跟踪或删除额外标头，则可以更新 Web 应用程序以允许跨域访问，也可以在脚本中启动 Chrome 浏览器时使用 `disable-web-security` 标志来禁用 Web 安全。

您可以覆盖 CloudWatch Synthetics 所用的启动参数，并通过使用 CloudWatch Synthetics 启动函数传递其他 `disable-web-security` 标记参数。有关更多信息，请参阅 可用于 Node.js 金丝雀脚本的库函数。

> **Note**
>
> 如果您使用的是 `syn-nodejs-2.1` 或更高运行时版本，则可以覆盖 CloudWatch Synthetics 所用的启动参数。

金丝雀争用情况问题

为了在使用 CloudWatch Synthetics 时获得最佳体验，请确保为金丝雀编写的代码具有幂等性。否则，在极少数情况下，当金丝雀在不同的运行中与相同资源交互时，Canary 运行可能会遇到争用情况。

排查 VPC 上金丝雀的问题

如果您在创建或更新使用 VPC 的金丝雀后遇到问题，以下其中一部分可能会帮助您解决问题。

新的金丝雀处于错误状态或无法更新金丝雀

如果您创建要在 VPC 上运行的金丝雀，并且它立即进入错误状态，或者您无法更新金丝雀以在 VPC 上运行，则金丝雀的角色可能没有正确的权限。要在 VPC 上运行，金丝雀必须具有权限 `ec2:CreateNetworkInterface`、`ec2:DescribeNetworkInterfaces` 和 `ec2:DeleteNetworkInterface`。这些权限包含在 `AWSLambdaVPCAccessExecutionRole` 托管策略中。有关更多信息，请参阅执行角色和用户权限。

如果在创建金丝雀时发生此问题，必须删除此金丝雀，然后创建一个新的金丝雀。如果您使用 CloudWatch 控制台创建新金丝雀，请在 Access Permissions（访问权限）下选择 Create a new role（创建新角色）。此时将创建一个新角色，该角色包含运行金丝雀所需的全部权限。

如果在更新金丝雀时发生此问题，可以再次更新金丝雀并提供具有必要权限的新角色。

“未返回测试结果”错误

如果金丝雀显示“未返回测试结果”错误，以下问题之一可能是导致这个问题的原因：

- 如果您的 VPC 不具有 Internet 访问权限，则必须使用 VPC 终端节点授予金丝雀对 CloudWatch 和 Amazon S3 的访问权限。您必须在 VPC 中启用 DNS 解析和 DNS 主机名选项，才能正确解析这些终端节点地址。有关更多信息，请参阅 将 DNS 与 VPC 配合使用 以及 将 CloudWatch 和 CloudWatch Synthetics 与接口 VPC 端点配合使用。
- 金丝雀必须在 VPC 内的私有子网中运行。要检查是否存在此问题，请在 VPC 控制台中打开子网页面。检查您在配置金丝雀时选择的子网。如果它们具有通往互联网网关 (igw-) 的路径，则不是私有子网。

要帮助您解决这些问题，请参阅金丝雀日志。

要查看来自金丝雀的日志事件，请执行以下操作：

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Log groups（日志组）。
3. 选择金丝雀日志组的名称。日志组名称以 `/aws/lambda/cwsyn-`*`canary-name`* 开头。

金丝雀脚本示例代码

本节包含对 CloudWatch Synthetics 金丝雀脚本的一些可能的函数进行说明的代码示例。

Node.js 和 Puppeteer 示例

设置 Cookie

网站依赖 Cookie 来提供自定义功能或跟踪用户。通过在 CloudWatch Synthetics 脚本中设置 Cookie，您可以模拟此自定义行为并对其进行验证。

例如，某个网站可能会对再次访问该网站的用户显示登录链接，而不是注册链接。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();

    // Set cookies.  I found that name, value, and either url or domain are required
 fields.
    const cookies = [{
      'name': 'cookie1',
      'value': 'val1',
      'url': url
    },{
      'name': 'cookie2',
      'value': 'val2',
      'url': url
    },{
      'name': 'cookie3',
```

```
      'value': 'val3',
      'url': url
    }];

    await page.setCookie(...cookies);

    // Navigate to the url
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
 30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'],
 timeout: timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " +
 JSON.stringify(cookiesSet));
    });

};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

设备仿真

您可以编写模拟各种设备的脚本，以便粗略估计页面在这些设备上的外观和行为。

以下示例模拟的是 iPhone 6 设备。有关仿真的更多信息，请参阅 Puppeteer 文档中的 page.emulate(options)。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];

    // INSERT URL here
    const URL = "https://amazon.com";

    let page = await synthetics.getPage();
```

```
    await page.emulate(iPhone);

    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }

    await page.waitFor(15000);

    await synthetics.takeScreenshot('loaded', 'loaded');

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

多步骤 API 金丝雀

此示例代码演示了一个具有两个 HTTP 步骤的 API 金丝雀：对正面和负面测试用例测试同一个 API。传递步骤配置以启用请求/响应标头报告。此外，其会隐藏 Authorization（授权）标头和 X-Amz-Security-Token，因为它们二者包含用户凭证。

当此脚本用作金丝雀时，您可以查看有关每个步骤和相关 HTTP 请求的详细信息，例如步骤通过/失败、持续时间和性能指标（如 DNS 查找时间和第一字节时间）。也可以查看金丝雀运行的 2xx、4xx 和 5xx 的数量。

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const apiCanaryBlueprint = async function () {

    // Handle validation for positive scenario
    const validatePositiveCase = async function(res) {
        return new Promise((resolve, reject) => {
```

```
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };

    // Handle validation for negative scenario
    const validateNegativeCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 400) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            resolve();
        });
    };

    let requestOptionsStep1 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };

    let headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep1['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
```

```
    // Change the configuration at global level or add as step configuration for
 individual steps
    let stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };


    await synthetics.executeHttpStep('Verify GET products API with valid name',
 requestOptionsStep1, validatePositiveCase, stepConfig);

    let requestOptionsStep2 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/canary/InvalidName(',
        'port': 443,
        'protocol': 'https:'
    };

    headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep2['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
    stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpStep('Verify GET products API with invalid name',
 requestOptionsStep2, validateNegativeCase, stepConfig);
```

```
};

exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Python 和 Selenium 示例

以下示例 Selenium 代码是一个金丝雀，当未加载目标元素时，此金丝雀会失败并显示自定义错误消息。

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
 load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

金丝雀和 X-Ray 跟踪

您可以选择对使用 `syn-nodejs-2.0` 或更高版本运行时的金丝雀启用活动 AWS X-Ray 跟踪。启用跟踪后，将针对由使用浏览器、AWS SDK 或 HTTP 或 HTTPS 模块的金丝雀发出的所有调用发送 X-Ray 跟踪。启用了跟踪功能的 Canary 会显示在 X-Ray 跟踪地图上，在您为应用程序启用此功能后显示在 Application Signals 中。

> **Note**
>
> 亚太地区（雅加达）尚不支持在金丝雀上激活 X-Ray 追踪。

当 Canary 出现在 X-Ray 跟踪地图上时，其会显示为新的客户端节点类型。您可以将鼠标悬停在某个金丝雀节点上以查看有关延迟、请求和故障的数据。您也可以选择该金丝雀节点以在页面底部查看更多数据。在页面的此区域，您可以选择 View in Synthetics（在 Synthetics 中查看）以跳转到 CloudWatch Synthetics 控制台，了解有关金丝雀的更多详细信息，或选择 View Traces（查看跟踪）来查看有关此金丝雀运行的跟踪的更多详细信息。

启用了跟踪的金丝雀的详细信息页面上还有一个 Tracing（跟踪）选项卡，其中包含有关金丝雀运行的跟踪和分段的详细信息。

启用跟踪会将金丝雀的运行时增加 2.5% 至 7%。

启用了跟踪的金丝雀必须使用具有以下权限的角色。如果您在创建金丝雀时使用控制台创建角色，则会为其授予这些权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Sid230934",
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments"
            ],
            "Resource": "*"
        }
    ]
}
```

金丝雀产生的跟踪会产生费用。有关 X-Ray 定价的更多信息，请参阅 AWS X-Ray 定价。

在 VPC 上运行金丝雀

您可以在 VPC 上的终端节点上，以及在内部公有终端节点上运行金丝雀。要在 VPC 上运行金丝雀，您必须在 VPC 上同时启用 DNS 解析和 DNS 主机名选项。有关更多信息，请参阅在您的 VPC 中使用 DNS。

当您在 VPC 终端节点上运行金丝雀时，必须提供一种将其指标发送到 CloudWatch 并将其构件发送到 Amazon S3 的方法。如果 VPC 已启用 Internet 访问，则您无需再执行任何操作。金丝雀将在您的 VPC 中执行，但可以访问 Internet 以上传其指标和构件。

如果 VPC 尚未启用互联网访问功能，您有两个选项：

- 启用 VPC 的互联网访问功能。有关更多信息，请参阅以下部分 为 VPC 上的金丝雀提供互联网访问权限。
- 如果您希望将 VPC 保持私有状态，可以将金丝雀配置为通过私有 VPC 终端节点将其数据发送到 CloudWatch 和 Amazon S3。如果您尚未这样做，则必须为 CloudWatch (com.amazonaws.*region*.monitoring) 创建 VPC 端点，并为 Amazon S3 创建网关端点。有关更多信息，请参阅 将 CloudWatch 和 CloudWatch Synthetics 与接口 VPC 终端节点结合使用 和适用于 Amazon S3 的 Amazon VPC 终端节点。

为 VPC 上的金丝雀提供互联网访问权限

请按照以下步骤为您的 VPC 金丝雀提供互联网访问权限，或者为您的 VPC 金丝雀分配静态 IP 地址

为 VPC 上的金丝雀提供互联网访问

1. 在 VPC 上的公有子网中创建 NAT 网关。有关说明，请参阅创建 NAT 网关。
2. 在启动金丝雀的私有子网中的路由表中添加新路由。指定以下内容：
   - 对于 Destination（目标），输入 **`0.0.0.0/0`**。
   - 对于目标，选择 NAT 网关，然后选择您创建的 NAT 网关的 ID。
   - 选择 Save routes（保存路由）。

   有关在路由表中添加路由的更多信息，请参阅在路由表中添加和删除路由。

> **Note**
>
> 确保通往 NAT 网关的路由处于活动状态。如果 NAT 网关被删除，但您尚未更新路由，则它们将处于黑洞状态。有关更多信息，请参阅使用 NAT 网关。

加密金丝雀构件

CloudWatch Synthetics 将屏幕截图、HAR 文件和报告等金丝雀构件存储在 Amazon S3 存储桶中。默认情况下，这些构件使用 AWS 托管式密钥加密。有关更多信息，请参阅客户密钥和 AWS 密钥。

您可以选择使用其他加密选项。CloudWatch Synthetics 支持以下选项：

- SSE-S3 – 使用 Amazon S3 托管式密钥进行服务器端加密 (SSE)。
- SSE-KMS – 使用 AWS KMS 客户托管式的密钥进行服务器端加密 (SSE)。

如果您想通过 AWS 托管式密钥使用默认加密选项，则不需要任何额外权限。

要使用 SSE-S3 加密，则需要将 SSE_S3 指定为创建或更新金丝雀时的加密模式。使用此加密模式不需要任何额外权限。有关更多信息，请参阅使用服务器端加密与 Amazon S3 托管式加密密钥 (SSE-S3) 保护数据。

要使用 AWS KMS 客户托管式密钥，则需要将 SSE-KMS 指定为创建或更新金丝雀时的加密模式，并提供密钥的 Amazon Resource Name (ARN)。您还可以使用跨账户 KMS 密钥。

要使用客户托管式密钥，需要以下设置：

- 您的金丝雀的 IAM 角色必须具有使用密钥加密构件的权限。如果您使用可视化监控，还必须授予其解密构件的权限。

```
{
    "Version": "2012-10-17",
    "Statement": {"Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "Your KMS key ARN"
    }
```

```
}
```

- 您可以将 IAM 角色添加到密钥策略中，而不是向 IAM 角色添加权限。如果您对多个金丝雀使用相同的角色，您应该考虑这种方法。

```
{
    "Sid": "Enable IAM User Permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "Your synthetics IAM role ARN"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

- 如果您使用的是跨账户 KMS 密钥，请参阅是否允许其他账户中的用户使用 KMS 密钥。

使用客户托管式密钥时查看加密的金丝雀构件

要查看金丝雀构件，请更新客户托管式密钥以向查看构件的用户提供 AWS KMS 解密权限。或者，向正在查看构件的用户或 IAM 角色添加解密权限。

默认 AWS KMS 策略让账户中的 IAM 策略能够允许访问 KMS 密钥。如果您使用的是跨账户 KMS 密钥，请参阅为什么跨账户用户在尝试访问由自定义 AWS KMS 密钥加密的 Amazon S3 对象时会出现“拒绝访问”的错误？

有关因 KMS 密钥而被拒绝访问的故障排除的更多信息，请参阅密钥访问故障排除。

使用可视化监控时更新构件位置和加密

为执行可视化监控，CloudWatch Synthetics 会将您的屏幕截图与在基准运行中获得的基准屏幕截图进行比较。如果您更新构件位置或加密选项，必须执行以下操作之一：

- 确保您的 IAM 角色对之前的 Amazon S3 位置和新的 Amazon S3 位置都具有足够的权限，以获取构件。同时，确保它对以前的和新的加密方法及 KMS 密钥都具有权限。
- 通过选择下一个金丝雀运行作为新基准来创建新基准。如果您使用此选项，则只需确保您的 IAM 角色对构件的新位置和加密选项具有足够的权限。

我们建议第二个选项，即选择下一次运行作为新基准。这避免了对之前用于金丝雀的构件位置或加密选项的依赖。

例如，假设您的金丝雀使用构件位置 A 和 KMS 密钥 K 来上传构件。如果您将金丝雀更新为构件位置 B 和 KMS 密钥 L，则可以确保您的 IAM 角色具有对两个构件位置（A 和 B）以及两个 KMS 密钥（K 和 L）的权限。或者，您可以选择下一次运行作为新基准，并确保您的金丝雀 IAM 角色具有对构件位置 B 和 KMS 密钥 L 的权限。

查看金丝雀统计数据和详细信息

您可以查看有关金丝雀的详细信息，并查看有关其运行的统计数据。

要能够查看有关金丝雀运行结果的所有详细信息，您必须登录具有足够权限的账户。有关更多信息，请参阅 CloudWatch 金丝雀的必需角色和权限。

查看金丝雀统计数据和详细信息

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。

   在有关您已创建的金丝雀的详细信息中：

   - Status (状态) 直观地显示了已通过其最新运行的金丝雀的数目。
   - Groups（组）显示了您创建的组，并显示其中有多少组存在故障或警报的金丝雀。
   - Slowest performers（最慢执行者）显示了金丝雀执行最慢的组和区域。这一数据是在选定的时间跨度内、将组或区域内所有金丝雀的平均持续时间相加，并除以组或区域中的金丝雀数量计算得出的。如果选择最慢组的指标，则会筛选该表，仅显示最慢组及其金丝雀。该表按平均持续时间排序。
   - 在页面底部附近有一张显示了所有金丝雀的表格。其中一列显示了为每个金丝雀创建的警报。仅显示符合金丝雀告警命名标准的告警。该标准为 `Synthetics-Alarm-`*`canaryName`*`-`*`index`*。您在 CloudWatch 控制台的 Synthetics 部分中创建的金丝雀告警会自动使用此命名约定。如果金丝雀告警是在 CloudWatch 控制台的 Alarms（告警）部分或通过使用 AWS CloudFormation 创建的，并且您未使用此命名约定，则告警会工作但不会出现在此列表中。
3. 要查看单个金丝雀的更多详细信息，请在 Canaries（金丝雀）表中选择该金丝雀的名称。

   在有关金丝雀的详细信息中：

   - Availability（可用性）选项卡显示了有关此金丝雀的最近运行的信息。

在 Canary runs (金丝雀运行) 下，您可以选择其中一条线，查看有关运行的详细信息。

在图表下，您可以选择 Steps（步骤）、Screenshot（屏幕截图）、Logs（日志）或 HAR file（HAR 文件）来查看这些类型的详细信息。如果金丝雀启用了活动跟踪，您还可以选择 Traces（跟踪）查看来自金丝雀运行的跟踪信息。

金丝雀运行的日志存储在 S3 存储桶和 CloudWatch Logs 中。

屏幕截图显示客户查看您的网页的情况。您可以使用 HAR 文件（HTTP 归档文件）查看有关网页性能的详细数据。您可以分析 Web 请求列表并捕获性能问题，例如某个项的加载时间。日志文件显示金丝雀运行与网页之间的交互记录，可用于确定错误的详细信息。

如果金丝雀使用的是 `syn-nodejs-2.0-beta` 或更高版本的运行时，则您可以按状态代码、请求大小或持续时间对 HAR 文件进行排序。

Steps（步骤）选项卡显示了金丝雀的步骤列表、每个步骤的状态、故障原因、步骤执行后的 URL、屏幕截图和步骤执行持续时间。对于具有 HTTP 步骤的 API 金丝雀，如果您使用的是 `syn-nodejs-2.2` 或更高版本的运行时，您可以查看步骤和相应的 HTTP 请求。

选择 HTTP Requests（HTTP 请求）选项卡查看金丝雀发出的每个 HTTP 请求的日志。您可以查看请求/响应标头、响应体、状态代码、错误和性能计时（总持续时间、TCP 连接时间、TLS 握手时间、第一字节时间和内容传输时间）。所有使用后台 HTTP/HTTPS 模块的 HTTP 请求都在此处捕获。

预设情况下，在 API 金丝雀中，出于安全原因，请求标头、响应标头、请求体和响应体不包含在报告中。如果您选择将它们包含在报告中，则这些数据仅存储在 S3 存储桶中。有关如何将此数据包括在报告中，请参阅 executeHttpStep(stepName, requestOptions, [callback], [stepConfig])。

支持文本、HTML 和 JSON 类型的响应体内容。支持文本/HTML、文本/纯文本、应用程序/JSON 和应用程序/x-amz-json-1.0 等内容类型。不支持经压缩的响应。

- Monitoring（监控）选项卡显示了由此金丝雀发布的 CloudWatch 指标的图表。有关这些指标的更多信息，请参阅 金丝雀发布的 CloudWatch 指标。

  在金丝雀发布的 CloudWatch 图表下方，是与金丝雀的 Lambda 代码相关的 Lambda 指标图表。
- Configuration（配置）选项卡显示了有关金丝雀的配置和计划信息。
- Groups（组）选项卡显示了与此金丝雀关联的组（如果有）。

- Tags（标签）选项卡显示了与金丝雀关联的标签。

金丝雀发布的 CloudWatch 指标

金丝雀会将以下指标发布到 CloudWatchSynthetics 命名空间中的 CloudWatch。有关查看 CloudWatch 指标的信息，请参阅 查看可用的指标。

指标	描述
SuccessPercent	此金丝雀成功运行且未发现任何故障的运行数百分比。 有效维度：CanaryName 有效统计数据：平均值 单位：百分比
Duration	金丝雀运行时长，以毫秒为单位。 有效维度：CanaryName 有效统计数据：平均值 单位：毫秒
Error	金丝雀运行其完整脚本失败的次数。 有效维度：CanaryName 有效统计数据：Sum
2xx	金丝雀执行的返回“OK”响应且响应代码介于 200 到 299 之间的网络请求数。 针对使用 syn-nodejs-2.0 或更高版本运行时的 UI 金丝雀和使用 syn-nodejs-2.2 或更高版本运行时的 API 金丝雀报告此指标。 有效维度：CanaryName 有效统计数据：Sum

指标	描述
	单位：计数
4xx	金丝雀执行的返回“Error”响应且响应代码介于 400 到 499 之间的网络请求数。 针对使用 syn-nodejs-2.0 或更高版本运行时的 UI 金丝雀和使用 syn-nodejs-2.2 或更高版本运行时的 API 金丝雀报告此指标。 有效维度：CanaryName 有效统计数据：Sum 单位：计数
5xx	金丝雀执行的返回“Fault”响应且响应代码介于 500 到 599 之间的网络请求数。 针对使用 syn-nodejs-2.0 或更高版本运行时的 UI 金丝雀和使用 syn-nodejs-2.2 或更高版本运行时的 API 金丝雀报告此指标。 有效维度：CanaryName 有效统计数据：Sum 单位：计数
Failed	执行失败的金丝雀运行数。这些失败与金丝雀本身有关。 有效维度：CanaryName 有效统计数据：Sum 单位：计数

指标	描述
Failed requests	金丝雀在目标网站上执行但失败且无响应的 HTTP 请求数。 有效维度：CanaryName 有效统计数据：Sum 单位：计数
VisualMonitoringSuccessPercent	在金丝雀运行期间成功匹配基准屏幕截图的可视化比较数百分比。 有效维度：CanaryName 有效统计数据：平均值 单位：百分比
VisualMonitoringTotalComparisons	在金丝雀运行期间发生的可视化比较总数。 有效维度：CanaryName 单位：计数

> **Note**
>
> 使用 Synthetics 库中 executeStep() 或 executeHttpStep() 方法的金丝雀还会发布 SuccessPercent 和 Duration 指标，指标包含每个步骤的 CanaryName 和 StepName 维度。

编辑或删除金丝雀脚本

您可以编辑或删除现有金丝雀。

编辑金丝雀

在编辑金丝雀时，即使您并未更改其计划，该计划也会在您编辑金丝雀时相应地重置。例如，如果您的金丝雀每小时运行，而您编辑了该金丝雀，则该金丝雀会在编辑完成后立即运行，然后每一小时运行一次。

编辑或更新金丝雀

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。
3. 选择金丝雀名称旁边的按钮，然后选择 Actions（操作）、Edit（编辑）。
4. （可选）如果此金丝雀执行屏幕截图可视化监控，并且您希望将金丝雀的下一次运行设置为基准，请选择 Set next run as new baseline（将下一次运行设置为新基准）。
5. （可选）如果此金丝雀执行屏幕截图可视化监控，并且您希望从可视化监控中删除某个屏幕截图，或者您希望在可视化比较期间指定要忽略部分屏幕截图，请在 Visual Monitoring（可视化监控）下选择 Edit Baseline（编辑基准）。

   此时会显示屏幕截图，您可以执行下列操作之一：

   - 若要删除用于可视化监控的屏幕截图，请选择 Remove screenshot from visual test baseline（从可视化测试基准中删除屏幕截图）。
   - 若要指定要在可视化对比过程中忽略的屏幕截图部分，请单击并拖动以绘制屏幕中要忽略的区域。针对要在对比中忽略的所有区域执行完成此操作后，选择 Save（保存）。
6. 对金丝雀做出任何其他更改，然后选择 Save（保存）。

删除金丝雀

当您删除金丝雀时，可以选择是否同时删除该金丝雀使用和创建的其他资源。当您删除金丝雀时，还应删除以下内容：

- 此金丝雀使用的 Lambda 函数和层。其前缀为 `cwsyn-`*`MyCanaryName`*。
- 为此金丝雀创建的 CloudWatch 告警。这些告警的名称都以 `Synthetics-Alarm-`*`MyCanaryName`* 开头。有关删除警报的更多信息，请参阅编辑或删除 CloudWatch 警报。
- Amazon S3 对象和存储桶，例如金丝雀的结果位置和构件位置。
- 为金丝雀创建的 IAM 角色。它们的名称为 `role/service-role/CloudWatchSyntheticsRole-`*`MyCanaryName`*。
- CloudWatch Logs 中为金丝雀创建的日志组。这些日志组具有以下名称：`/aws/lambda/cwsyn-`*`MyCanaryName-randomId`*。

在删除金丝雀之前，您可能需要查看金丝雀详细信息并记下此信息。这样，您就可以在删除金丝雀后删除正确的资源。

删除金丝雀

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。
3. 如果金丝雀当前处于 RUNNING 状态，则您必须停止它。仅可删除处于 STOPPED、READY(NOT_STARTED) 或 ERROR 状态的金丝雀。

   要停止金丝雀，请选择金丝雀名称旁边的按钮，然后选择 Actions（操作）、Stop（停止）。
4. 选择金丝雀名称旁边的按钮，然后选择 Actions（操作）、Delete（删除）。
5. 选择是否同时删除为该金丝雀创建和供其使用的其他资源。这包括 Lambda 函数和层，以及该金丝雀的 IAM 角色和 IAM 策略。

   要删除该金丝雀的 IAM 角色和 IAM 策略，您必须拥有足够的权限。有关更多信息，请参阅 用于 CloudWatch Synthetics 的 AWS 托管式（预定义）策略。
6. 在框中输入 **Delete**，然后选择 Delete（删除）。
7. 删除为金丝雀所用和为其创建的其他资源，如本节前面所列内容。

启动、停止、删除或更新多个金丝雀脚本的运行时

您可以通过一个操作停止、启动、删除或更新多达五个金丝雀脚本的运行时。如果您更新金丝雀脚本的运行时，它会更新为适用于金丝雀脚本所用语言和框架的最新运行时。

如果您选择了多个金丝雀脚本，并且只有其中一部分处于对所选操作有效的状态，则该操作仅在该操作有效的金丝雀脚本上执行。例如，如果您选择了一些当前正在运行的金丝雀脚本和一些未运行的金丝雀脚本，然后选择启动金丝雀脚本，则尚未运行的金丝雀脚本将会启动，而已经运行的金丝雀脚本不会受到影响。

如果您选择的所有金丝雀脚本都不适用于某项操作，则该操作在菜单中不可用。

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Synthetics 金丝雀。
3. 选中要停止、启动或删除的金丝雀脚本旁边的复选框。
4. 选择 Actions（操作），然后选择 Start（启动）、Stop（停止）、Delete（删除）或 Update Runtime（更新运行时）。

使用 Amazon EventBridge 监控金丝雀事件

在金丝雀更改状态或完成运行时，Amazon EventBridge 事件规则可以通知您。EventBridge 提供近乎实时的系统事件流，这些系统事件描述的是 AWS 资源发生的变化。CloudWatch Synthetics 尽最大努力将这些事件发送到 EventBridge。尽最大努力发送事件是指 CloudWatch Synthetics 会尝试将所有事件发送到 EventBridge，但在某些极少数情况下，可能无法发送事件。EventBridge 至少会处理所接收的所有事件一次。此外，事件侦听器可能无法按事件的发生顺序接收事件。

> **Note**
>
> Amazon EventBridge 是一种事件总线服务，您可以用其轻松地将应用程序与来自各种来源的数据相连接。有关更多信息，请参阅 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？。

CloudWatch Synthetics 会在金丝雀改变状态或完成运行时发出事件。您可以创建一个包含事件模式的 EventBridge 规则，以匹配从 CloudWatch Synthetics 发送的所有事件类型，或者仅匹配特定事件类型。当金丝雀触发规则时，EventBridge 会调用规则中定义的目标操作。这样，您就可以发送通知、捕获事件信息以及采取纠正操作，以响应金丝雀的状态更改或完成运行。例如，您可以为以下使用案例创建规则：

- 在金丝雀运行失败时执行调查
- 在金丝雀变为 ERROR 状态时执行调查
- 跟踪金丝雀的生命周期
- 作为工作流程的一部分监控金丝雀运行成功或故障

CloudWatch Synthetics 发送的事件示例

本节列出了 CloudWatch Synthetics 发送的事件示例。有关事件格式的更多信息，请参阅 EventBridge 中的事件和事件模式。

金丝雀状况更改

在此事件类型中，`current-state` 和 `previous-state` 的值可以是：

`CREATING | READY | STARTING | RUNNING | UPDATING | STOPPING | STOPPED | ERROR`

```
{
                "version": "0",
```

```
                "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",
                "detail-type": "Synthetics Canary Status Change",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:19:43Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "current-state": "STOPPED",
                                "previous-state": "UPDATING",
                                "source-location": "NULL",
                                "updated-on": 1612909161.767,
                                "changed-config": {
                                                "executionArn": {
                                                                "previous-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                },
                                                "vpcId": {
                                                                "current-value": "NULL"
                                                },
                                                "testCodeLayerVersionArn": {
                                                                "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                }
                                },
                                "message": "Canary status has changed"
                }
}
```

已完成的成功金丝雀运行

```
{
```

```
                "version": "0",
                "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
                "detail-type": "Synthetics Canary TestRun Successful",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:01Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-28ddbe266797/2021/02/09/22/23-41-200",
                                "test-run-status": "PASSED",
                                "state-reason": "null",
                                "canary-run-timeline": {
                                                "started": 1612909421,
                                                "completed": 1612909441
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

已完成的失败金丝雀运行

```
{
                "version": "0",
                "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
                "detail-type": "Synthetics Canary TestRun Failure",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:27Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "0df3823e-7e33-4da1-8194-
b04e4d4a2bf6",
```

```
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
                                "test-run-status": "FAILED",
                                "state-reason": "\"Error: net::ERR_NAME_NOT_RESOLVED
 \""
                                "canary-run-timeline": {
                                                "started": 1612909461,
                                                "completed": 1612909467
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

可能是事件重复或者顺序颠倒。要确定事件的顺序，请使用 `time` 属性。

创建 EventBridge 规则的先决条件

在为 CloudWatch Synthetics 创建 EventBridge 规则之前，您应执行以下操作：

- 熟悉 EventBridge 中的事件、规则和目标。
- 创建和配置由 EventBridge 规则调用的目标。规则可以调用许多类型的目标，包括：
  - Amazon SNS 主题
  - AWS Lambda 函数
  - Kinesis Streams
  - Amazon SQS 队列

有关更多信息，请参阅 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？和 Amazon EventBridge 入门。

创建 EventBridge 规则 (CLI)

以下示例中的步骤将创建一条 EventBridge 规则，该规则会在 `us-east-1` 中名为 `my-canary-name` 的金丝雀完成一次运行或更改状态时发布 Amazon SNS 主题。

1. 创建 规则。

```
aws events put-rule \
  --name TestRule \
  --region us-east-1 \
```

```
  --event-pattern "{\"source\": [\"aws.synthetics\"], \"detail\": {\"canary-name\":
[\"my-canary-name\"]}}"
```

模式中省略的属性都将会被忽略。

2. 将主题添加为规则目标。

   - 将 *topic-arn* 替换为 Amazon SNS 主题的 Amazon Resource Name (ARN)。

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> 要允许 Amazon EventBridge 调用您的目标主题，您必须将基于资源的策略添加到您的主题中。有关更多信息，请参阅 Amazon EventBridge 用户指南中的 Amazon SNS 权限。

有关更多信息，请参阅 Amazon EventBridge 用户指南中的 EventBridge 中的事件和事件模式。

CloudWatch RUM

借助 CloudWatch RUM，可执行真实用户监控，从实际用户会话中近乎实时地收集和查看有关 Web 应用程序性能的客户端数据。可视化和可分析的数据包括页面加载时间、客户端错误和用户行为。当查看这些数据时，可以看到所有数据汇总在一起，也可以查看按客户使用的浏览器和设备划分的细分数据。

您可以使用收集的数据快速识别和调试客户端性能问题。CloudWatch RUM 可帮助您可视化应用程序性能中的异常，并查找相关的调试数据，例如错误消息、堆栈跟踪和用户会话。还可以使用 RUM 了解终端用户的影响范围，包括用户数量、地理位置和使用的浏览器。

为 CloudWatch RUM 收集的终端用户数据将保留 30 天，然后自动删除。如果要将 RUM 事件保留更长时间，可以选择让应用程序监控将事件的副本发送到账户中的 CloudWatch Logs。然后，便可以调整该日志组的保留期。

要使用 RUM，需要创建应用程序监控并提供一些信息。RUM 会生成一个可粘贴到应用程序中的 JavaScript 代码段。该代码段拉入了 RUM Web 客户端代码。RUM Web 客户端从应用程序中一定百分

比的用户会话中捕获数据，这些数据显示在预构建的控制面板中。您可以指定要从中收集数据的用户会话百分比。

CloudWatch RUM 与 Application Signals 集成，可以发现和监控您的应用程序服务、客户端、Synthetics Canary 和服务依赖项。使用 Application Signals 查看您的服务列表或可视地图，根据您的服务级别目标（SLO）查看运行状况指标，并深入查看关联 X-Ray 跟踪以便更详细地进行问题排查。要在 Application Signals 中查看 RUM 客户端页面请求，请通过创建应用程序监控或手动配置 RUM Web 客户端打开 X-Ray 活动跟踪。您的 RUM 客户端显示在连接到您的服务的服务地图以及所调用服务的服务详细信息页面上。

RUM Web 客户端是开源的。有关更多信息，请参阅 CloudWatch RUM Web 客户端。

性能注意事项

本节讨论使用 CloudWatch RUM 时的性能注意事项。

- 负载性能影响 – CloudWatch RUM Web 客户端可作为 JavaScript 模块被安装到您的 Web 应用程序当中，或从内容分发网络（CDN）异步加载到 Web 应用程序。这不会阻止应用程序的加载过程。CloudWatch RUM 旨在不对应用程序的加载时间产生可感知的影响。
- 运行时影响 – RUM Web 客户端执行处理以记录 RUM 数据并将其分派到 CloudWatch RUM 服务。由于事件不频繁且处理量很小，因此 CloudWatch RUM 旨在不对应用程序的性能产生可检测的影响。
- 网络影响 – RUM Web 客户端定期向 CloudWatch RUM 服务发送数据。应用程序运行期间会定期分派数据，也会在浏览器卸载应用程序之前立即分派数据。浏览器卸载应用程序之前立即发送的数据将作为信标发送，该信标旨在不对应用程序的卸载时间产生可检测的影响。

RUM 定价

使用 CloudWatch RUM 时，CloudWatch RUM 收到的每个 RUM 事件可能都会收费。使用 RUM Web 客户端收集的每个数据项目都视为 RUM 事件。RUM 事件示例包括页面视图、JavaScript 错误和 HTTP 错误。您可以选择各应用程序监控收集的事件类型。您可以激活或停用选项来收集性能遥测事件、JavaScript 错误、HTTP 错误和 X-Ray 跟踪。有关选择这些选项的更多信息，请参阅 步骤 2：创建应用程序监控 和 CloudWatch RUM Web 客户端收集的信息。有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

区域可用性

CloudWatch RUM 目前在以下区域中可用：

- 美国东部（弗吉尼亚州北部）
- 美国东部（俄亥俄州）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈州）
- 非洲（开普敦）
- 亚太地区（雅加达）
- 亚太地区（孟买）
- 亚太地区（海得拉巴）
- 亚太地区（墨尔本）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 中东（巴林）
- 中东（阿联酋）
- 南美洲（圣保罗）

主题

- 使用 CloudWatch RUM 的 IAM 策略
- 设置应用程序以使用 CloudWatch RUM

- 配置 CloudWatch RUM Web 客户端
- 区域化
- 使用页面组
- 指定自定义元数据
- 发送自定义事件
- 查看 CloudWatch RUM 控制面板
- 您可以使用 CloudWatch RUM 收集的 CloudWatch 指标
- CloudWatch RUM 的数据保护和数据隐私
- CloudWatch RUM Web 客户端收集的信息
- 管理使用 CloudWatch RUM 的应用程序
- CloudWatch RUM 配额
- CloudWatch RUM 故障排除

使用 CloudWatch RUM 的 IAM 策略

为了完全管理 CloudWatch RUM，必须以具有 AmazonCloudWatchRUMFullAccess IAM 策略的 IAM 用户或角色身份登录。此外，可能需要其他策略或权限：

- 要创建应用程序监控来创建新的 Amazon Cognito 身份池进行授权，需要具备 Admin IAM 角色或 AdministratorAccess IAM 策略。
- 要创建将数据发送到 CloudWatch Logs 的应用程序监控，必须登录具有以下权限的 IAM 角色或策略：

```
{
    "Effect": "Allow",
    "Action": [
        "logs:PutResourcePolicy"
    ],
    "Resource": [
        "*"
    ]
}
```

需要查看 CloudWatch RUM 数据但不需要创建 CloudWatch RUM 资源的其他用户，可以授予 AmazonCloudWatchRUMReadOnlyAccess 策略。

设置应用程序以使用 CloudWatch RUM

使用本章节中的步骤设置应用程序，以开始使用 CloudWatch RUM 从真实用户会话中收集性能数据。

主题

- 步骤 1：授权应用程序将数据发送到 AWS
- 步骤 2：创建应用程序监控
- （可选）步骤 3：手动修改代码段以配置 CloudWatch RUM Web 客户端
- 步骤 4：将代码段插入应用程序
- 步骤 5：通过生成用户事件测试应用程序监控设置

步骤 1：授权应用程序将数据发送到 AWS

要使用 CloudWatch RUM，应用程序必须获得授权。

您有三种设置授权的选项：

- 用 CloudWatch RUM 为应用程序创建新的 Amazon Cognito 身份池。这是最简单的设置方法。这是默认选项。

  身份池将包含未经验证的身份。这允许 CloudWatch RUM Web 客户端将数据发送到 CloudWatch RUM，而无需对应用程序的用户进行身份验证。

  Amazon Cognito 身份池具有附加的 IAM 角色。Amazon Cognito 未经验证的身份允许 Web 客户端担任 IAM 角色，此角色已获得授权，可向 CloudWatch RUM 发送数据。
- 使用现有的 Amazon Cognito 身份池。在这种情况下，还必须修改附加到身份池的 IAM 角色。对于支持未经身份验证用户的身份池，请使用此选项。您只能使用同一区域中的身份池。
- 使用已设置的现有身份提供商的身份验证。在这种情况下，您必须从身份提供商处获取凭证，并且应用程序必须将这些凭证转发到 RUM Web 客户端。

  对于仅支持经身份验证用户的身份池，请使用此选项。

以下各节包含了这些选项的更多详细信息。

CloudWatch RUM 创建新的 Amazon Cognito 身份池

这是最简单的设置选项，如果选择此选项，则无需其他设置步骤。您必须拥有管理权限才能使用此选项。有关更多信息，请参阅 使用 CloudWatch RUM 的 IAM 策略。

通过此选项，CloudWatch RUM 将创建以下资源：

- 新的 Amazon Cognito 身份池
- 未经身份验证的 Amazon Cognito 身份。这允许 RUM Web 客户端在不验证应用程序用户的情况下担任 IAM 角色。
- RUM Web 客户端将担任的 IAM 角色。附加到此角色的 IAM 策略允许其使用带有应用程序监控资源的 `PutRumEvents` API。换句话说，其允许 RUM Web 客户端向 RUM 发送数据。

RUM Web 客户端使用 Amazon Cognito 身份获取 AWS 凭证。AWS 凭证与 IAM 角色关联。IAM 角色有权通过 AppMonitor 资源使用 `PutRumEvents`。

Amazon Cognito 会发送必要的安全令牌，以确保应用程序能够向 CloudWatch RUM 发送数据。CloudWatch RUM 生成的 JavaScript 代码段包括以下代码行以启用身份验证。

```
   {
    identityPoolId: [identity pool id],  // e.g., 'us-west-2:EXAMPLE4a-66f6-4114-902a-
EXAMPLEbad7'
   }
);
```

使用现有的 Amazon Cognito 身份池

如果您选择使用现有的 Amazon Cognito 身份池，则需要在将应用程序添加到 CloudWatch RUM 时指定身份池。身份池必须支持访问未经验证的身份。您只能使用同一区域中的身份池。

您还必须将以下权限添加到已附加到与此身份池关联的 IAM 角色的 IAM 策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountid]:appmonitor/[app monitor
 name]"
        }
    ]
}
```

Amazon Cognito 会发送必要的安全令牌，以确保应用程序能够访问 CloudWatch RUM。

第三方提供商

如果您选择使用第三方提供商的私有身份验证，则必须从身份提供商处获取凭证并将其转发到 AWS。执行此操作的最佳方法是使用安全令牌供应商。您可以使用任意安全令牌供应商，包括使用 AWS Security Token Service 的 Amazon Cognito。有关 AWS STS 的更多信息，请参阅欢迎使用 AWS Security Token Service API 参考。

如果您想在这种情况下使用 Amazon Cognito 作为令牌供应商，则可以配置 Amazon Cognito，以与身份验证提供商结合使用。有关更多信息，请参阅 Amazon Cognito 身份池入门（联合身份）。

配置 Amazon Cognito 以与身份提供商结合使用后，还需要执行以下操作：

- 创建具有以下权限的 IAM 角色。应用程序将使用该角色访问 AWS。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:[region]:[accountID]:appmonitor/[app monitor
 name]"
        }
    ]
}
```

- 将以下内容添加到应用程序中，让应用程序将提供商的凭证传递给 CloudWatch RUM。插入该行，以便在用户登录到应用程序并且应用程序收到用于访问 AWS 的凭证后运行。

```
cwr('setAwsCredentials', {/* Credentials or CredentialProvider */});
```

有关 AWS JavaScript SDK 凭证提供商的更多信息，请参阅 SDK for JavaScript v3 开发者指南中的在 Web 浏览器中设置凭证、SDK for JavaScript v2 开发人员指南中的在 Web 浏览器中设置凭证以及 @aws-sdk/凭证提供商。

您还可以使用适用于 CloudWatch RUM Web 客户端的软件开发工具包来配置 Web 客户端身份验证方法。有关 Web 客户端软件开发工具包的更多信息，请参阅 CloudWatch RUM Web 客户端软件开发工具包。

步骤 2：创建应用程序监控

要在应用程序中开始使用 CloudWatch RUM，您可以创建应用程序监控。创建应用程序监控后，RUM 会生成一个可粘贴到应用程序中的 JavaScript 代码段。该代码段拉入了 RUM Web 客户端代码。RUM Web 客户端从应用程序一定百分比的用户会话中捕获数据并将其发送到 RUM。

创建应用程序监控

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 Add app monitor（添加应用程序监控）。
4. 输入应用程序的信息和设置：

   - 在 App monitor name（应用程序监控名称）中，输入在 CloudWatch RUM 控制台中用于识别此应用程序监控的名称。
   - 在 Application domain（应用程序域）中，输入应用程序具有管理权限的顶级域名。必须采用 URL 域格式。

     选择 Include sub domain（包括子域），让应用程序监控也能从顶级域名下的所有子域中收集数据。
5. 在 Configure RUM data collection（配置 RUM 数据收集）中，指定是否希望应用程序监控收集以下每项内容：

   - Performance telemetry（性能遥测）– 收集有关页面加载和资源加载时间的信息
   - JavaScript errors（JavaScript 错误）– 收集应用程序引发的未处理 JavaScript 错误的信息
   - HTTP errors（HTTP 错误）– 收集应用程序引发的 HTTP 错误的信息

   选择这些选项可以提供有关应用程序的更多信息，但也会生成更多 CloudWatch RUM 事件，从而产生更多的费用。

   如果未选择以上任何选项，应用程序监控仍会收集会话开启事件和页面 ID，以便您可以查看有多少用户正在使用应用程序，包括按操作系统类型和版本、浏览器类型和版本、设备类型和位置的细分。
6. 如果您希望能够从采样的用户会话中收集用户 ID 和会话 ID，选择 Check this option to allow the CloudWatch RUM Web Client to set cookies（选中此选项以允许 CloudWatch RUM Web 客户端设置 Cookie）。用户 ID 由 RUM 随机生成。有关更多信息，请参阅 CloudWatch RUM Web 客户端 Cookie（或类似技术）。

7. 在 Session samples（会话样本）中，输入将用于收集 RUM 数据的用户会话百分比。默认值为 100%。减少此数字将提供更少的数据，但可以降低费用。有关 RUM 定价的更多信息，请参阅 RUM 定价。
8. 为 CloudWatch RUM 收集的终端用户数据将保留 30 天，然后删除。如果您想在 CloudWatch Logs 中保留 RUM 事件的副本并配置这些副本的保留时间，请选择 Data storage（数据存储）下的 Check this option to store your application telemetry data in your CloudWatch Logs account（选中此选项以将应用程序遥测数据存储在 CloudWatch Logs 账户中）。预设情况下，CloudWatch Logs 日志组会将数据保留 30 天。您可以在 CloudWatch Logs 控制台中调整保留期。
9. 在 Authorization（授权）中，指定是使用新的或现有的 Amazon Cognito 身份池，还是使用其他身份提供商。创建新的身份池是最简单的选择，无需其他设置步骤。有关更多信息，请参阅 步骤 1：授权应用程序将数据发送到 AWS。

   创建新的 Amazon Cognito 身份池需要管理权限。有关更多信息，请参阅 使用 CloudWatch RUM 的 IAM 策略。
10. （可选）预设情况下，您将 RUM 代码段添加到应用程序时，Web 客户端会将 JavaScript 标签注入到应用程序所有页面的 HTML 代码中以监控使用情况。要更改此选项，请选择 Configure pages（配置页面），然后选择 Include only these pages（仅包含这些页面）或 Exclude these pages（排除这些页面）。然后指定要包含或排除的页面。要指定包含或排除的页面，请输入其完整 URL。要指定其他页面，请选择 Add URL（添加 URL）。
11. 要启用 AWS X-Ray 跟踪应用程序监控采样的用户会话，选择 Active tracing（活动跟踪），然后选择 Trace my service with AWS X-Ray（使用跟踪我的服务）。

    如果选择此项，将会跟踪在应用程序监控采样用户会话期间发出的 `XMLHttpRequest` 和 `fetch` 请求。然后，您可在 RUM 控制面板、X-Ray 跟踪地图和跟踪详细信息页面中查看来自这些用户会话的跟踪和分段。为应用程序启用相应功能后，这些用户会话还将作为客户端页面显示在 Application Signals 中。

    通过对 CloudWatch RUM Web 客户端进行其他配置更改，您可以向 HTTP 请求添加 X-Ray 跟踪标头，以启用到下游 AWS 托管式服务的用户会话端到端跟踪。有关更多信息，请参阅 启用 X-Ray 端到端跟踪。
12. （可选）要向应用程序监控添加标签，请选择 Tags（标签）、Add new tag（添加新标签）。

    然后，对于 Key（键），输入标签的名称。您可以在 Value (值) 中添加可选的标签值。

    要添加其他标签，请再次选择 Add new tag（添加新标签）。

有关更多信息，请参阅标记 AWS 资源。

13. 选择 Add app monitor（添加应用程序监控）。
14. 在 Sample code（示例代码）部分，您可以复制要使用的代码段以添加到您的应用程序当中。我们建议您选择 JavaScript 或 TypeScript，使用 NPM 安装 CloudWatch RUM Web 客户端，并将其作为 JavaScript 模块。

    或者，您可以选择 HTML 来使用内容分发网络（CDN），以便安装 CloudWatch RUM Web 客户端。使用 CDN 的缺点在于，Web 客户端常被广告拦截器拦截。
15. 选择 Copy（复制）或 Download（下载），然后选择 Done（完成）。

（可选）步骤 3：手动修改代码段以配置 CloudWatch RUM Web 客户端

您可以在代码段插入应用程序之前对其进行修改，以激活或停用多个选项。有关更多信息，请参阅 CloudWatch RUM Web 客户端文档。

正如本章节中所述，您必须注意三个配置选项。

防止收集可能包含个人信息的资源 URL

预设情况下，CloudWatch RUM Web 客户端配置为记录应用程序下载的资源的 URL。这些资源包括 HTML 文件、图像、CSS 文件、JavaScript 文件等。对于某些应用程序，URL 可能包含个人身份信息（PII）。

如果应用程序属于这种情况，则强烈建议您通过在代码段配置中设置 `recordResourceUrl: false` 以禁用收集资源 URL，然后再将其插入到应用程序中。

手动记录页面浏览次数

默认情况下，Web 客户端会在页面首次加载以及调用浏览器的历史 API 时记录页面浏览次数。默认页面 ID 为 `window.location.pathname`。但在某些情况下，您可能需要覆盖此行为并利用应用程序以编程方式记录页面浏览量。这样您可以控制页面 ID 及其记录时间。例如，假设一个 Web 应用程序的 URI 带有变量标识符，例如 `/entity/123` 或 `/entity/456`。默认情况下，CloudWatch RUM 会为每个具有与路径名匹配的不同页面 ID 的 URI 生成页面浏览事件，但您可能希望改用相同的页面 ID 对它们进行分组。为此，请使用 `disableAutoPageView` 配置禁用 Web 客户端的页面浏览自动化，然后使用 `recordPageView` 命令设置所需的页面 ID。有关更多信息，请参阅 GitHub 上的特定于应用程序的配置。

嵌入式脚本示例：

```
cwr('recordPageView', { pageId: 'entityPageId' });
```

JavaScript 模块示例：

```
awsRum.recordPageView({ pageId: 'entityPageId' });
```

启用 X-Ray 端到端跟踪

在创建应用程序监控时，选择 Trace my service with AWS X-Ray（使用跟踪我的服务）将启用跟踪在应用程序监控采样的用户会话期间发出的 `XMLHttpRequest` 和 `fetch` 请求。然后，您可在 CloudWatch RUM 控制面板、X-Ray 跟踪地图和跟踪详细信息页面查看来自这些 HTTP 请求的跟踪。

默认情况下，这些客户端跟踪未连接到下游服务器端跟踪。要将客户端跟踪连接到服务器端跟踪并启用端到端跟踪，请在 Web 客户端中将 `addXRayTraceIdHeader` 选项设置为 `true`。这将导致 CloudWatch RUM Web 客户端向 HTTP 请求添加 X-Ray 跟踪标头。

下面的代码块展示了添加客户端跟踪的示例。为提高可读性，此示例中省略了一些配置选项。

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            enableXRay: true,
            telemetries: [
                'errors',
                'performance',
                [ 'http', { addXRayTraceIdHeader: true } ]
            ]
        }
    );
</script>
```

> **Warning**
>
> 如果使用 SigV4 签署请求，则将 CloudWatch RUM Web 客户端配置为向 HTTP 请求添加 X-Ray 跟踪标头可能会导致跨源资源共享 (CORS) 失败或该请求的签名无效。有关更多信息，请

参阅 CloudWatch RUM Web 客户端文档。我们强烈建议您在生产环境中添加客户端 X-Ray 跟踪标头之前测试应用程序。

有关更多信息，请参阅 CloudWatch RUM Web 客户端文档

步骤 4：将代码段插入应用程序

接下来，将上一部分中创建的代码段插入到应用程序中。

> **Warning**
>
> 由代码段下载和配置的 Web 客户端使用 Cookie（或类似技术）来帮助您收集终端用户数据。在插入代码段之前，请参阅 在控制台中按元数据属性筛选。

如果您没有以前生成的代码段，则可以按照 如何查找已生成的代码段？ 中的说明进行查找。

将 CloudWatch RUM 代码段插入应用程序

1. 将您在上一部分中复制或下载的代码段插入应用程序的 <head> 元素。在 <body> 元素或任何其他 <script> 标签之前插入。

   以下是一个已生成的代码段示例：

```
<script>
(function (n, i, v, r, s, c, x, z) {
    x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
    window[n] = function (c, p) {
        x.q.push({c: c, p: p});
    };
    z = document.createElement('script');
    z.async = true;
    z.src = s;
    document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
    '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
    '1.0.0',
    'us-east-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        sessionSampleRate: 1,
```

```
        identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cfd21443",
        endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        enableXRay: false
    });
</script>
```

2. 如果应用程序是多页 Web 应用程序，则必须对要包含在数据集合中的每个 HTML 页面重复步骤 1。

步骤 5：通过生成用户事件测试应用程序监控设置

插入代码段并且已更新的应用程序正在运行之后，您可以通过手动生成用户事件对其进行测试。要进行测试，建议您执行以下操作。此测试将产生标准 CloudWatch RUM 费用。

- 在 Web 应用程序中页面之间导航。
- 使用不同的浏览器和设备创建多个用户会话。
- 提出请求。
- 导致 JavaScript 错误。

生成一些事件后，请在 CloudWatch RUM 控制面板中查看。有关更多信息，请参阅 查看 CloudWatch RUM 控制面板。

来自用户会话的数据可能需要 15 分钟才能在控制面板中显示。

如果在应用程序中生成事件 15 分钟后没有看到数据，请参阅 CloudWatch RUM 故障排除。

配置 CloudWatch RUM Web 客户端

应用程序可以使用 CloudWatch RUM 生成的其中一个代码段来安装 CloudWatch RUM Web 客户端。生成的代码段支持两种安装方法：作为 JavaScript 模块并通过 NPM，或通过内容分发网络（CDN）。为了获得最佳性能，我们建议使用 NPM 安装方法。有关使用此方法的更多信息，请参阅作为 JavaScript 模块安装。

如果您使用 CDN 安装选项，广告拦截器可能拦截由 CloudWatch RUM 提供的默认 CDN。若用户安装了广告拦截器，它将禁用应用程序监控。因此，我们建议您只在第一次启动 CloudWatch RUM 时使用默认的 CDN。有关此问题缓解方法的更多信息，请参阅检测应用程序。

代码段位于 HTML 文件的 <head> 标签，并且下载 Web 客户端来完成其安装，然后为其监控的应用程序配置 Web 客户端。该代码段是一种自动执行的函数，类似于以下内容。在此示例中，为提高可读性，已省略代码段函数的正文。

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(
'cwr',
'00000000-0000-0000-0000-000000000000',
'1.0.0',
'us-west-2',
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
{ /* Configuration Options Here */ }
);
<script>
```

参数

代码段接受六个参数：

- 用于在 Web 客户端上运行命令的命名空间，例如 `'cwr'`
- 应用程序监控的 ID，例如 `'00000000-0000-0000-0000-000000000000'`
- 应用程序版本，例如 `'1.0.0'`
- 应用程序监控的 AWS 区域，例如 `'us-west-2'`
- Web 客户端的 URL，例如 `'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'`
- 特定于应用程序的配置选项。有关更多信息，请参阅以下 章节。

忽略错误

CloudWatch RUM Web 客户端会侦听应用程序中发生的所有类型的错误。如果您的应用程序发出了 JavaScript 错误，而您不想在 CloudWatch RUM 控制面板中查看这些错误，则可以配置 CloudWatch RUM Web 客户端以筛选出这些错误，以便在 CloudWatch RUM 控制面板上仅看到相关的错误事件。例如，您可能选择不在控制面板中查看某些 JavaScript 错误，因为您已经确定了这些错误的修复方法，而且这些错误的数量掩盖了其他错误。您可能还想忽略那些您无法修复的错误，因为它们归第三方所有的库所有。

有关如何分析 Web 客户端以筛选出特定 JavaScript 错误的更多信息，请参阅 Web 客户端 Github 文档中的错误中的示例。

配置选项

有关 CloudWatch RUM Web 客户端可用的配置选项信息，请参阅 CloudWatch RUM Web 客户端文档

区域化

本节说明了在不同区域的应用程序中使用 CloudWatch RUM 的策略。

我的 Web 应用程序部署在多个 AWS 区域

如果您的 Web 应用程序部署在多个 AWS 区域，您有三种选择：

- 在一个区域、一个账户中部署一个应用程序监视器，为所有区域提供服务。
- 在唯一的账户中为每个区域部署单独的应用程序监视器。
- 为每个地区部署单独的应用程序监视器，并且都在一个账户中。

使用一个应用程序监视器的优势在于，所有数据都将集中到一个可视化效果中，并且所有日志都写入 CloudWatch Logs 中的同一个日志组。使用单个应用程序监视器时，请求会有少量的额外延迟，并且会出现单点故障。

使用多个应用程序监视器可以消除单点故障，但可以防止将所有数据合并为一个可视化效果。

CloudWatch RUM 尚未在我的应用程序部署的某些区域中启动

CloudWatch RUM 已在许多地区推出，具有广泛的地理覆盖范围。通过在 CloudWatch RUM 可用的区域设置 CloudWatch RUM，您可以获得好处。如果您在终端用户所连接的区域中设置了应用程序监视器，则终端用户可以随时随地访问他们的会话。

然而，CloudWatch RUM 尚未在 AWS GovCloud（美国东部）、AWS GovCloud（美国西部）或中国任何地区推出。您无法从这些区域向 CloudWatch RUM 发送数据。

使用页面组

使用页面组将应用程序中的不同页面相互关联，以便您可以查看页面组的汇总分析。例如，您可能希望查看所有登录页面的汇总页面加载时间。

您可以在 CloudWatch RUM Web 客户端中向页面查看事件添加一个或多个标签，从而将页面放入页面组中。以下示例将 `/home` 页面放入名为 `en` 的页面组和名为 `landing` 的页面组。

嵌入式脚本示例

```
cwr('recordPageView', { pageId: '/home', pageTags: ['en', 'landing']});
```

JavaScript 模块示例

```
awsRum.recordPageView({ pageId: '/home', pageTags: ['en', 'landing']});
```

> **Note**
>
> 页面组旨在促进对不同页面进行聚合分析。有关如何定义和操作应用程序的 `pageIds` 的信息，请参阅（可选）步骤 3：手动修改代码段以配置 CloudWatch RUM Web 客户端 中的手动记录页面浏览量部分。

指定自定义元数据

CloudWatch RUM 将额外的数据作为元数据附加到每个事件。事件元数据由键值对形式的属性组成。您可以使用这些属性在 CloudWatch RUM 控制台中搜索或筛选事件。默认情况下，CloudWatch RUM 会为您创建一些元数据。有关默认元数据的更多信息，请参阅 RUM 事件元数据。

您还可以使用 CloudWatch RUM Web 客户端将自定义元数据添加到 CloudWatch RUM 事件中。自定义元数据可以包括会话属性和页面属性。

要添加自定义元数据，必须使用 1.10.0 或更高版本的 CloudWatch RUM Web 客户端。

要求和语法

每个事件可以在元数据中包含多达 10 个自定义属性。自定义属性的语法要求如下：

- 键
  - 最多 128 个字符
  - 可以包含字母数字字符、冒号（:）和下划线（_）
  - 不能以 `aws:` 开头。
  - 不能完全由以下列出的任何保留关键字组成。可以将这些关键字用作较长键名的一部分。
- 值
  - 最多 256 个字符

- 必须是字符串、数字或布尔值

保留关键字

您不能将以下保留关键字用作完整的键名。您可以使用以下关键字作为较长键名的一部分，例如 `applicationVersion`。

- `browserLanguage`
- `browserName`
- `browserVersion`
- `countryCode`
- `deviceType`
- `domain`
- `interaction`
- `osName`
- `osVersion`
- `pageId`
- `pageTags`
- `pageTitle`
- `pageUrl`
- `parentPageId`
- `platformType`
- `referrerUrl`
- `subdivisionCode`
- `title`
- `url`
- `version`

> **Note**
>
> 如果属性包含无效的键或值，或者已经达到每个事件 10 个自定义属性的限制，则 CloudWatch RUM 会从 RUM 事件中删除自定义属性。

添加会话属性

如果您配置自定义会话属性，这些属性会添加到会话的所有事件中。您可以在 CloudWatch RUM Web 客户端初始化期间或运行时使用 `addSessionAttributes` 命令配置会话属性。

例如，您可以将应用程序版本添加为会话属性。然后，在 CloudWatch RUM 控制台中，您可以按版本筛选错误，以确定错误率增加是否与应用程序的特定版本有关。

在初始化期间添加会话属性，NPM 示例

粗体代码部分添加了会话属性。

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';

try {
  const config: AwsRumConfig = {
    allowCookies: true,
    endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
    guestRoleArn: "arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth",
    identityPoolId: "us-west-2:00000000-0000-0000-0000-000000000000",
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance'],
    sessionAttributes: {
        applicationVersion: "1.3.8"
    }
  };

  const APPLICATION_ID: string = '00000000-0000-0000-0000-000000000000';
  const APPLICATION_VERSION: string = '1.0.0';
  const APPLICATION_REGION: string = 'us-west-2';

  const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    config
  );
} catch (error) {
  // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

在运行时添加会话属性，NPM 示例

```
awsRum.addSessionAttributes({
    applicationVersion: "1.3.8"
})
```

在初始化期间添加会话属性，嵌入式脚本示例

粗体代码部分添加了会话属性。

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            sessionSampleRate:1,
            guestRoleArn:'arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth',
            identityPoolId:'us-west-2:00000000-0000-0000-0000-000000000000',
            endpoint:'https://dataplane.rum.us-west-2.amazonaws.com',
            telemetries:['errors','http','performance'],
            allowCookies:true,
            sessionAttributes: {
                applicationVersion: "1.3.8"
            }
        }
    );
</script>
```

在运行时添加会话属性，嵌入式脚本示例

```
<script>
    function addSessionAttribute() {
        cwr('addSessionAttributes', {
            applicationVersion: "1.3.8"
        })
    }

</script>
```

添加页面属性

如果您配置自定义页面属性，这些属性会添加到当前页面上的所有事件中。您可以在 CloudWatch RUM Web 客户端初始化期间或运行时使用 `recordPageView` 命令配置页面属性。

例如，您可以将页面模板添加为页面属性。然后，在 CloudWatch RUM 控制台中，您可以按页面模板筛选错误，以确定错误率增加是否与应用程序的特定页面模板有关。

在初始化期间添加页面属性，NPM 示例

粗体代码部分添加了页面属性。

```
const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    { disableAutoPageView:  true // optional }
);
awsRum.recordPageView({
    pageId:'/home',
    pageAttributes: {
      template: 'artStudio'
    }
});
const credentialProvider = new CustomCredentialProvider();
if(awsCreds) awsRum.setAwsCredentials(credentialProvider);
```

在运行时添加页面属性，NPM 示例

```
awsRum.recordPageView({
    pageId: '/home',
    pageAttributes: {
        template: 'artStudio'
    }
});
```

在初始化期间添加页面属性，嵌入式脚本示例

粗体代码部分添加了页面属性。

```
<script>
```

```
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            disableAutoPageView: true //optional
        }
    );
    cwr('recordPageView', {
       pageId: '/home',
       pageAttributes: {
           template: 'artStudio'
       }
    });
    const awsCreds = localStorage.getItem('customAwsCreds');
    if(awsCreds) cwr('setAwsCredentials', awsCreds)
</script>
```

在运行时添加页面属性，嵌入式脚本示例

```
<script>
    function recordPageView() {
        cwr('recordPageView', {
            pageId: '/home',
            pageAttributes: {
                template: 'artStudio'
            }
        });
    }
</script>
```

在控制台中按元数据属性筛选

要使用任何内置或自定义元数据属性在 CloudWatch RUM 控制台中筛选可视化内容，请使用搜索栏。在搜索栏中，您可以通过 key=value 的形式指定多达 20 个筛选词用于可视化内容。例如，如果要仅筛选 Chrome 浏览器的数据，可以添加筛选词 browserName=Chrome。

默认情况下，CloudWatch RUM 控制台会检索 100 个最常见的属性键和值，并将它们显示在搜索栏的下拉列表中。要添加更多元数据属性作为筛选词，请在搜索栏中输入完整的属性键和值。

一个筛选条件可以包含多达 20 个筛选词，每个应用程序监视器最多可以保存 20 个筛选条件。当您保存筛选条件时，筛选条件会保存在 Saved filters（保存的筛选条件）下拉列表中。您还可以删除保存的筛选条件。

发送自定义事件

CloudWatch RUM 可记录并摄取 CloudWatch RUM Web 客户端收集的信息 中列出的事件。如果您使用 1.12.0 或更高版本的 CloudWatch RUM Web 客户端，则您可以定义、记录和发送其他自定义事件。对于您定义的每个事件类型，您可以定义事件类型名称和要发送的数据。每个自定义事件负载最多可达 6KB。

只有当应用程序监视器启用了自定义事件时，才会摄取自定义事件。要更新应用程序监视器的配置设置，请使用 CloudWatch RUM 控制台或 UpdateAppMonitor API。

启用自定义事件，然后定义和发送自定义事件，即可搜索它们。要搜索自定义事件，请使用 CloudWatch RUM 控制台中的 Events（事件）选项卡。请使用事件类型进行搜索。

要求和语法

自定义事件由事件类型和事件详细信息组成。其要求如下：

- 事件类型
  - 这可以是事件的 type（类型）或 name（名称）。例如，名为 JsError 的 CloudWatch RUM 内置事件类型的事件类型为 `com.amazon.rum.js_error_event`。
  - 长度必须介于 1-256 个字符之间。
  - 可以是字母数字字符、下划线、连字符和句点的组合。
- 事件详细信息
  - 包含要在 CloudWatch RUM 中记录的实际数据。
  - 必须是由字段和值组成的对象。

记录自定义事件的示例

有两种方法可以在 CloudWatch RUM Web 客户端中记录自定义事件。

- 使用 CloudWatch RUM Web 客户端的 `recordEvent` API。
- 使用自定义插件。

使用 **`recordEvent`** API 发送自定义事件，NPM 示例

```
awsRum.recordEvent('my_custom_event', {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
)
```

使用 **recordEvent** API 发送自定义事件，嵌入式脚本示例

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

使用自定义插件发送自定义事件的示例

```
// Example of a plugin that listens to a scroll event, and
// records a 'custom_scroll_event' that contains the timestamp of the event.
class MyCustomPlugin implements Plugin {
    // Initialize MyCustomPlugin.
    constructor() {
        this.enabled;
        this.context;
        this.id = 'custom_event_plugin';
    }
    // Load MyCustomPlugin.
    load(context) {
        this.context = context;
        this.enable();
    }
```

```
    // Turn on MyCustomPlugin.
    enable() {
        this.enabled = true;
        this.addEventHandler();
    }
    // Turn off MyCustomPlugin.
    disable() {
        this.enabled = false;
        this.removeEventHandler();
    }
    // Return MyCustomPlugin Id.
    getPluginId() {
        return this.id;
    }
    // Record custom event.
    record(data) {
        this.context.record('custom_scroll_event', data);
    }
    // EventHandler.
    private eventHandler = (scrollEvent: Event) => {
        this.record({timestamp: Date.now()})
    }
    // Attach an eventHandler to scroll event.
    private addEventHandler(): void {
        window.addEventListener('scroll', this.eventHandler);
    }
    // Detach eventHandler from scroll event.
    private removeEventHandler(): void {
        window.removeEventListender('scroll', this.eventHandler);
    }
}
```

查看 CloudWatch RUM 控制面板

CloudWatch RUM 可帮助您从用户会话中收集有关应用程序性能的数据，包括页面加载时间、Apdex 分数、使用的浏览器和设备、用户会话的地理位置以及存在错误的会话。所有这些信息都显示在控制面板中。

查看 RUM 控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。

Overview（概览）选项卡显示您创建的其中一个应用程序监控收集的信息。

窗格的顶行显示此应用程序监控的以下信息：

- 页面加载数
- 页面平均加载速度
- Apdex 分数
- 与应用程序监控相关的任何告警的状态

应用程序性能索引 (Apdex) 分数表示终端用户的满意度。分数范围为 0（最不满意）到 1（最满意）。分数仅基于应用程序性能。不要求用户对应用程序进行评级。有关 Apdex 分数的更多信息，请参阅 CloudWatch RUM 如何设置 Apdex 分数。

其中一些窗格包含可用于进一步检查数据的链接。选择这些链接中的任何一个都将显示详细视图，视图顶部为性能、错误、HTTP 请求、会话、事件浏览器和设备和用户历程选项卡。

3. 要进一步聚焦，请选择 List view（列表视图）选项卡，然后选择要聚焦的应用程序监控名称。这将显示所选应用程序监控的以下选项卡。

   - Performance（性能）选项卡显示页面性能信息，包括加载时间、会话信息、请求信息、Web 重要信息和一段时间内的页面加载次数。此视图包含在 Page loads（页面加载）、Requests（请求）和 Location（位置）焦点间切换视图的控制件。
   - 错误选项卡显示 Javascript 错误信息，包括用户最常看到的错误信息以及错误最多的设备和浏览器。此视图包括错误的直方图和错误的列表视图。您可以按用户和事件详细信息筛选错误列表。选择一条错误消息可查看更多详细信息。
   - HTTP 请求选项卡显示 HTTP 请求信息，包括错误最多的请求 URL 以及错误最多的设备和浏览器。此选项卡包括请求的直方图、请求的列表视图以及网络错误的列表视图。您可以按用户和事件详细信息筛选列表。选择响应代码或错误消息，以分别查看有关请求或网络错误的更多详细信息。
   - 会话选项卡显示会话指标。此选项卡包括会话启动事件的直方图和会话的列表视图。您可以按事件类型、用户详细信息和事件详细信息筛选会话列表。选择 sessionID 可查看有关会话的更多详细信息。
   - 事件选项卡显示 RUM 事件的直方图和事件的列表视图。您可以按事件类型、用户详细信息和事件详细信息筛选事件列表。选择 RUM 事件可查看原始事件。
   - Browsers & Device（浏览器和设备）选项卡显示访问应用程序的不同浏览器和设备的性能和使用情况等信息。此视图包括在浏览器和设备焦点间切换视图的控件。

如果将范围缩小到单个浏览器，则会看到按浏览器版本细分的数据。

- User Journey（用户历程）选项卡显示客户用于导航应用程序的路径。您可以看到客户进入应用程序的位置以及从应用程序退出的页面。您还可以看到其所用的路径以及遵循这些路径的客户百分比。您可以在节点上暂停以获取有关该页面的更多详细信息。您可以选择单个路径来突出显示连接以便更轻松查看。

4. （可选）在前六个选项卡中的任何一个选项卡上，可以选择页面按钮，并从列表中选择一个页面或页面组。这会将显示的数据缩小到应用程序的单个页面或页面组。您还可以将列表中的页面和页面组标记为收藏。

CloudWatch RUM 如何设置 Apdex 分数

应用程序性能索引 (Apdex) 是一种开放标准，其定义了报告、基准和评估应用程序响应时间的方法。Apdex 分数可帮助您了解和识别随时间推移对应用程序性能的影响。

Apdex 分数表示终端用户的满意度分数，分数范围为 0（最不满意）到 1（最满意度）。分数仅基于应用程序性能。不要求用户对应用程序进行评级。

每个 Apdex 分数均属于三个阈值之一。根据 Apdex 阈值和实际应用程序响应时间，有三种性能，如下所示：

- Satisfied（满意）– 实际应用程序响应时间小于等于 Apdex 阈值。对于 CloudWatch RUM，此阈值为 2000 毫秒或更低。
- Tolerable（尚可）– 实际应用程序响应时间大于 Apdex 阈值，但小于或等于 Apdex 阈值的四倍。对于 CloudWatch RUM，此阈值为 2000–8000 毫秒。
- Frustrating（不满）– 实际应用程序响应时间大于 Apdex 阈值的四倍。对于 CloudWatch RUM，此阈值为大于 8000 毫秒。

使用以下公式计算 0-1 Apdex 总分数：

```
(positive scores + tolerable scores/2)/total scores * 100
```

您可以使用 CloudWatch RUM 收集的 CloudWatch 指标

本节的表格列出了使用 CloudWatch RUM 自动收集的指标。您可以在 CloudWatch 控制台中查看这些指标。有关更多信息，请参阅 查看可用的指标。

您也可以选择向 CloudWatch 或 CloudWatch Evidently 发送扩展指标。有关更多信息，请参阅 扩展指标。

这些指标在名为 AWS/RUM 的指标命名空间中发布。所有以下指标通过 application_name 维度发布。此维度的值为应用程序监控的名称。一些指标还会通过其他维度发布，如表中所列。

指标	单位	描述
HttpStatusCodeCount	计数	应用程序中 HTTP 响应的计数，按响应状态代码显示。 其他维度： • event_details.response.status 是响应状态代码，例如 200、400、404 等。 • event_type 事件类型。目前，该维度的唯一可能值是 http。
Http4xxCount	计数	应用程序中 HTTP 响应的计数，响应状态代码为 4xx。 这些是根据会生成 4xx 代码的 http_event RUM 事件计算得出的。
Http5xxCount	计数	应用程序中 HTTP 响应的计数，响应状态代码为 5xx。

指标	单位	描述
		这些是根据会生成 5xx 代码的 http_event RUM 事件计算得出的。
JsErrorCount	计数	摄入的 JavaScript 错误事件的计数。
NavigationFrustratedTransaction	计数	比不满阈值 (8000ms) 高出 duration 的导航事件的计数。导航事件的持续时间在 PerformanceNavigationDuration 指标中跟踪。
NavigationSatisfiedTransaction	计数	duration 低于 Apdex 目标 (2000ms) 的导航事件的计数。导航事件的持续时间在 PerformanceNavigationDuration 指标中跟踪。
NavigationToleratedTransaction	计数	duration 在 2000ms 和 8000ms 之间的导航事件的计数。导航事件的持续时间在 PerformanceNavigationDuration 指标中跟踪。

指标	单位	描述
PageViewCount	计数	应用程序监控摄取的页面查看事件的计数。 该计数通过 page_view_event RUM 事件的计数计算得出。
PerformanceResourceDuration	毫秒	资源事件的 duration。 其他维度： • event_details.file.type 是资源事件的文件类型，例如样式表、文档、图像、脚本或字体。 • event_type 事件类型。目前，该维度的唯一可能值是 resource。
PerformanceNavigationDuration	毫秒	导航事件的 duration。
RumEventPayloadSize	字节	CloudWatch RUM 摄入的每个事件的规模。您还可以使用此指标的 SampleCount 统计数据来监控应用程序监控摄入的事件数量。

指标	单位	描述
SessionCount	计数	应用程序监控摄入的会话启动事件的计数。即，已启动的新会话的数量。
WebVitalsCumulativeLayoutShift	无	跟踪累计布局转移事件的值。
WebVitalsFirstInputDelay	毫秒	跟踪第一个输入延迟事件的值。
WebVitalsLargestContentfulPaint	毫秒	跟踪最大内容绘制事件的值。

可以向 CloudWatch 和 CloudWatch Evidently 发送的自定义指标和扩展指标

默认情况下，RUM 应用程序监视器会向 CloudWatch 发送指标。这些默认指标和维度列在可以使用 CloudWatch RUM 收集的 CloudWatch 指标中。

您还可以设置应用程序监控，以导出指标。应用程序监控可以发送扩展指标、自定义指标或两者。它可以将这些指标发送到 CloudWatch、CloudWatch Evidently 或两者。

- 自定义指标 – 自定义指标是由您定义的指标。有了自定义指标，您可以使用任何指标名称和命名空间。要派生指标，您可以使用任何自定义事件、内置事件、自定义属性或默认属性。

  您可以向 CloudWatch 或 CloudWatch Evidently 发送扩展指标。
- 扩展指标 – 允许您向 CloudWatch Evidently 发送默认 CloudWatch RUM 指标，以用于 Evidently 实验。您还可以将任何默认的 CloudWatch RUM 指标发送给具有其他维度的 CloudWatch。这样，这些指标可以为您提供更精细的视图。

主题

- 自定义指标
- 扩展指标

自定义指标

要发送自定义指标，您必须使用 AWS API 或 AWS CLI 而不是控制台。有关使用 AWS API 的信息，请参阅 PutRumMetricsDestination 和 BatchCreateRumMetricDefinitions。

一个目标可以包含的扩展指标定义和自定义指标定义的最大数量是 2000。对于您发送到每个目标的每个扩展指标，维度名称和维度值的每个组合都计入此限制。这也算作定价的 CloudWatch 自定义指标。

以下示例演示了如何创建从自定义事件派生的自定义指标。以下是使用的自定义事件示例：

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

根据此自定义事件，您可以创建一个自定义指标，用于计算 Chrome 浏览器访问该 amazonaws.com URL 的次数。以下定义在 RUM/CustomMetrics/PageVisits 命名空间中创建一个在您的账户中名为 AmazonVisitsCount 的指标。

```
{
    "AppMonitorName":"customer-appMonitor-name",
    "Destination":"CloudWatch",
    "MetricDefinitions":[
        {
            "Name":"AmazonVisitsCount",
            "Namespace":"PageVisit",
            "ValueKey":"event_details.visit_count",
            "UnitLabel":"Count",
            "DimensionKeys":{
                "event_details.current_url": "URL"
            },
            "EventPattern":"{\"metadata\":{\"browserName\":[\"Chrome\"]},\"event_type
\":[\"my_custom_event\"],\"event_details\": {\"current_url\": [\"amazonaws.com\"]}}"
        }
    ]
```

```
}
```

扩展指标

如果您设置了扩展指标，则可以执行以下一项或两项操作：

- 向 CloudWatch Evidently 发送默认的 CloudWatch RUM 指标，以用于 Evidently 实验。只有 PerformanceNavigationDuration、PerformanceResourceDuration、WebVitalsCumulativeLayoutShift、We 和 WebVitalsLargestContentfulPaint 指标可以发送到 Evidently。
- 将任何默认 CloudWatch RUM 指标发送到具有其他维度的 CloudWatch，以便这些指标为您提供更精细的视图。例如，您可以查看用户使用的特定浏览器特有的指标，或特定地理位置的用户的指标。

有关默认 CloudWatch RUM 指标的更多信息，请参阅 您可以使用 CloudWatch RUM 收集的 CloudWatch 指标。

一个目标可以包含的扩展指标定义和自定义指标定义的最大数量是 2000。对于您发送到每个目标的每个扩展指标或自定义指标，维度名称和维度值的每个组合都算作此限制的扩展指标。这也算作定价的 CloudWatch 自定义指标。

向 CloudWatch 发送扩展指标时，您可以使用 CloudWatch RUM 控制台为它们创建 CloudWatch 告警。

扩展指标作为 CloudWatch 自定义指标收费。有关更多信息，请参阅Amazon CloudWatch 定价。

对于应用程序监视器可以发送的所有指标名称，扩展指标支持以下维度。您可以使用 CloudWatch RUM 收集的 CloudWatch 指标 中列出了这些指标名称。

- `BrowserName`

  维度值示例：`Chrome`、`Firefox`、`Chrome Headless`
- `CountryCode` 使用 ISO-3166 格式，带有两个字母的代码。

  维度值示例：`US`、`JP`、`DE`
- `DeviceType`

  维度值示例：`desktop`、`mobile`、`tablet`、`embedded`
- `FileType`

  维度值示例：`Image`、`Stylesheet`

- OSName

  维度值示例：Linux、Windows, 、iOS、Android
- PageId

使用控制台设置扩展指标

要使用控制台向 CloudWatch 发送扩展指标，请执行以下步骤。

要向 CloudWatch Evidently 发送扩展指标，您必须使用 AWS API 或 AWS CLI 而不是控制台。有关使用 AWS API 向 CloudWatch 或 Evidently 发送扩展指标的信息，请参阅 PutRumMetricsDestination 和 BatchCreateRumMetricDefinitions。

要使用控制台设置应用程序监视器并向 CloudWatch 发送 RUM 扩展指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图），然后选择要发送指标的应用程序监视器名称。
4. 选择 Configuration（配置）选项卡，然后选择 RUM extended metrics（RUM 扩展指标）。
5. 选择 Send metrics（发送指标）。
6. 选择一个或多个与其他维度一起发送的指标名称。
7. 选择一个或多个因子用作这些指标的维度。当您做出选择时，您选择创建的扩展指标数量将显示在 Number of extended metrics（扩展指标数量）中。

   该数字的计算方法是将所选指标名称的数量乘以您创建的不同维度的数量。该数字表示您需要为多少自定义指标付费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

   a. 要发送以页面 ID 作为维度的指标，请选择 Browse for page ID（浏览页面 ID），然后选择要使用的页面 ID。
   b. 要发送以设备类型作为维度的指标，请选择 Desktop devices（桌面设备）或 Mobile and tablets（移动设备和平板电脑）。
   c. 要发送以操作系统作为维度的指标，请在 Operating system（操作系统）下选择一个或多个操作系统。
   d. 要发送以浏览器类型作为维度的指标，请在 Browsers（浏览器）下选择一个或多个浏览器。
   e. 要发送以地理位置作为维度的指标，请在 Locations（位置）下选择一个或多个位置。

      只有此应用程序监视器报告指标的位置才会出现在列表中供您选择。

8. 完成选择后，选择 Send metrics（发送指标）。
9. （可选）在 Extended metrics（扩展指标）列表中，如要创建监视其中一个指标的告警，请在该指标行中选择 Create alarm（创建告警）。

   有关 CloudWatch 告警的一般信息，请参阅 使用 Amazon CloudWatch 告警。有关为 CloudWatch RUM 扩展指标设置告警的教程，请参阅 教程：创建扩展指标并为其设置告警。

停止发送扩展指标

使用控制台停止发送扩展指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图），然后选择要发送指标的应用程序监视器名称。
4. 选择 Configuration（配置）选项卡，然后选择 RUM extended metrics（RUM 扩展指标）。
5. 选择一个或多个指标名称和维度组合以停止发送。然后选择 Actions（操作）、Delete（删除）。

教程：创建扩展指标并为其设置告警

本教程将演示如何设置要发送到 CloudWatch 的扩展指标，以及如何为该指标设置告警。在本教程中，您将创建一个指标来跟踪 Chrome 浏览器上的 JavaScript 错误。

设置该扩展指标并为其设置告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图），然后选择要发送该指标的应用程序监视器名称。
4. 选择 Configuration（配置）选项卡，然后选择 RUM extended metrics（RUM 扩展指标）。
5. 选择 Send metrics（发送指标）。
6. 选择 JSErrorCount。
7. 在 Browsers（浏览器）下，选择 Chrome。

   这种 JSErrorCount 和 Chrome 的组合将向 CloudWatch 发送一个扩展指标。该指标仅计算使用 Chrome 浏览器的用户会话的 JavaScript 错误。指标名称将是 JSErrorCount，维度名称将是 Browser（浏览器）。

8. 选择 Send metrics（发送指标）。
9. 在 Extended metrics（扩展指标）列表中，在 Name（名称）下显示 JSErrorCount 并在 BrowserName（浏览器名称）下显示 Chrome 的行中选择 Create alarm（创建告警）。
10. 在 Specify metric and conditions（指定指标和条件）下，确认 Metric name（指标名称）和 BrowserName（浏览器名称）字段已预先填充正确的值。
11. 对于 Statistic（统计数据），选择要用于告警的统计数据。对于这种类型的计数指标，Average（平均值）是一个不错的选择。
12. 对于时段，选择 5 分钟。
13. 在 Conditions（条件）下，执行以下操作：

    - 选择 Static（静态）。
    - 选择 Greater（大于），指定当错误数高于您要指定的阈值时告警应进入 ALARM 状态。
    - 在 than... 下，输入告警阈值数字。当 5 分钟内的错误数超过此数字时，告警将进入 ALARM 状态。
14. （可选）默认情况下，一旦错误数在 5 分钟内超过您设置的阈值数字，告警就会进入 ALARM 状态。您也可以选择更改此设置，使告警仅在超过此数字的时间多于一个 5 分钟周期时才进入 ALARM 状态。

    为此，请选择 Additional configuration（其他配置），然后为 Datapoints to alarm（触发告警的数据点数）指定错误数超过阈值的时间需要达到多少个 5 分钟周期才能触发告警。例如，您可以选择 2 选 2，仅在连续两个 5 分钟周期超过阈值时触发告警；您也可以选择 3 选 2，当连续三个 5 分钟周期中的任意两个超过阈值时触发告警。

    有关此类告警评估的更多信息，请参阅 评估告警。
15. 选择下一步。
16. 对于 Configure actions（配置操作），指定当告警进入 ALARM 状态时应进行的操作。要使用 Amazon SNS 接收通知，请执行以下操作：

    - 选择 Add notification（添加通知）。
    - 选择告警中。
    - 选择一个现有的 SNS 主题或创建一个新主题。如果您创建了一个新主题，请为其指定一个名称，并向其添加至少一个电子邮件地址。
17. 选择下一步。
18. 输入告警的名称和描述（可选），然后选择 Next（下一步）。
19. 检查详细信息，然后选择 Create alarm（创建告警）。

CloudWatch RUM 的数据保护和数据隐私

AWS 责任共担模式适用于 Amazon CloudWatch RUM 中的数据保护和数据隐私。如该模式中所述，AWS 负责保护运行所有 AWS 云的全球基础设施。您负责维护对托管在此基础架构上的内容的控制。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全博客上的 AWS 责任共担模式和 GDPR 博客文章。有关遵守 GDPR 要求的更多资源，请参阅一般数据保护条例 (GDPR) 中心。

Amazon CloudWatch RUM 会根据对要收集的终端用户数据的输入生成一个代码段，可将其嵌入到网站或 Web 应用程序代码中。由代码段下载和配置的 Web 客户端使用 Cookie（或类似技术）来帮助您收集终端用户数据。Cookie（或类似技术）的使用受某些司法管辖区的数据隐私法规约束。在使用 Amazon CloudWatch RUM 之前，强烈建议您评估适用法律规定的合规性义务，包括要求您针对 Cookie 的使用和终端用户数据的处理（包括收集）提供合法的隐私声明，并获得任何必要同意的任何适用法律要求。有关 Web 客户端如何使用 Cookie（或类似技术）以及 Web 客户端所收集的终端用户数据的更多信息，请参阅 CloudWatch RUM Web 客户端收集的信息 和 CloudWatch RUM Web 客户端 Cookie（或类似技术）。

强烈建议您不要在自由格式字段中输入敏感标识信息（例如终端用户客户账户号码、邮箱地址或其他个人信息）。您输入到 Amazon CloudWatch RUM 或其他服务中的任何数据都可能包含在诊断日志中。

CloudWatch RUM Web 客户端 Cookie（或类似技术）

默认情况下，CloudWatch RUM Web 客户端会收集有关用户会话的某些数据。您可以选择启用 Cookie，让 Web 客户端收集在页面加载间持续存在的用户 ID 和会话 ID。用户 ID 由 RUM 随机生成。

如果启用了这些 Cookie，当您查看此应用程序监控的 RUM 控制面板时，RUM 可以显示以下类型的数据。

- 基于用户 ID 的汇总数据，例如唯一用户数量和出错的不同用户数量。
- 基于会话 ID 的汇总数据，例如会话数量和出错的会话数量。
- 用户历程，每个采样用户会话所包含的页面序列。

> **Important**
>
> 如果不启用这些 Cookie（或类似技术），Web 客户端仍会记录有关终端用户会话的某些信息，例如浏览器类型/版本、操作系统类型/版本、设备类型等。收集这些信息是为了提供特定

于页面的洞察，例如 Web 重要信息、页面浏览次数和出错页面。有关所记录数据的更多信息，请参阅 CloudWatch RUM Web 客户端收集的信息。

CloudWatch RUM Web 客户端收集的信息

本节将记录 PuTrumEvent 架构，其定义了可以使用 CloudWatch RUM 从用户会话中收集的数据结构。

PuTrumEvent 请求将包含以下字段的数据结构发送到 CloudWatch RUM。

- 此批次 RUM 事件的 ID
- 应用程序监控详细信息，包括以下内容：
  - 应用程序监控 ID
  - 受监控的应用程序版本
- 用户详细信息，包括以下内容。只有应用程序监控启用 Cookie 时才会收集此信息。
  - Web 客户端生成的用户 ID
  - 会话 ID
- 此批次中的 RUM 事件数组。

RUM 活动架构

每个 RUM 事件的结构均包含以下字段。

- 事件 ID
- 时间戳
- 事件类型
- 用户代理
- Metadata
- RUM 事件详细信息

RUM 事件元数据

元数据包括页面元数据、用户代理元数据、地理位置元数据和域元数据。

页面元数据

页面元数据包括以下内容：

- 页面 ID
- 页面标题
- 父页面 ID。– 只有应用程序监控启用 Cookie 时才会收集此信息。
- 交互深度 – 只有在应用程序监控启用 Cookie 时才会收集此信息。
- 页面标签 – 您可以向页面事件添加标签，将页面组合在一起。有关更多信息，请参阅 使用页面组。

用户代理元数据

用户代理元数据包括以下内容：

- 浏览器语言
- 浏览器名称
- 浏览器版本
- 操作系统名称
- 操作系统版本
- 设备类型
- 平台类型

地理位置元数据

地理位置元数据包括以下内容：

- 国家/地区代码
- 细分代码

域元数据

域元数据包括 URL 域。

RUM 事件详细信息

根据事件类型，事件的详细信息遵循以下类型的架构之一。

会话开启事件

此事件不包含字段。只有应用程序监控启用 Cookie 时才会收集此信息。

页面视图架构

Page view（页面视图）事件包含以下属性。您可以通过配置 Web 客户端来停用页面视图收集。有关更多信息，请参阅 CloudWatch RUM Web 客户端文档。

名称	Type	描述
页面 ID	String	在应用程序中唯一代表此页面的 ID。默认情况下，这是 URL 路径。
父页面 ID	String	用户导航到当前页面时所在页面的 ID。只有应用程序监控启用 Cookie 时才会收集此信息。
交互深度	String	只有应用程序监控启用 Cookie 时才会收集此信息。

JavaScript 错误架构

代理生成的 JavaScript 错误事件包含以下属性。仅当您选择收集错误遥测时，Web 客户端才会收集这些事件。

名称	Type	描述
错误类型	String	错误名称（如果存在）。有关更多信息，请参阅 Error.prototype.name。 某些浏览器可能不支持错误类型。
错误消息	String	错误的消息。有关更多信息，请参阅 Error.prototype.message。如果错误字段不存在，则这是错误事件消息。有关更多信息，请参阅 ErroRevent。 不同浏览器之间的错误消息可能不一致。
堆栈跟踪	String	错误的堆栈跟踪（如果存在）将被截断为 150 个字符。有关更多信息，请参阅 Error.prototype.stack。

名称	Type	描述
		某些浏览器可能不支持堆栈跟踪。

DOM 事件架构

代理生成的文档对象模型 (DOM) 事件包含以下属性。默认情况下，不会收集这些事件。只有在激活交互遥测时才会收集。有关更多信息，请参阅 CloudWatch RUM Web 客户端文档。

名称	Type	描述
Event (事件)	String	DOM 事件的类型，例如单击、滚动或悬停。有关更多信息，请参阅事件参考。
元素	String	DOM 元素类型
元素 ID	String	如果生成事件的元素有 ID，则此属性存储该 ID。有关更多信息，请参阅 Element.id。
CSSLocator	String	用于识别 DOM 元素的 CSS 定位器。
InteractionId	String	用户与 UI 之间交互的唯一 ID。

导航事件架构

只有在应用程序监控激活性能遥测时，才会收集导航事件。

导航事件使用 Navigation timing Level 1 和 Navigation timing Level 2 API。并非所有浏览器都支持 Level 2 API，因此这些较新的字段是可选的。

> **Note**
>
> 时间戳指标基于 DOMHighResTimestamp。使用 Level 2 API，默认情况下，所有时间都相对于 `startTime`。但是对于 Level 1，需从时间戳指标中减去 `navigationStart` 指标以获取相对值。所有时间戳值均以毫秒为单位。

导航事件包含以下属性。

名称	Type	描述	注意
initiatorType	String	表示启动性能事件的资源类型。	值：“navigation” Level 1：“navigation” Level 2：entryData.initiatorType
navigationType	String	表示导航类型。 此属性非必需。	值：该值必须是以下之一： • navigate 是通过选择链接、在浏览器的地址栏中输入 URL、表单提交或通过脚本操作的启动而开始的导航，而不是通过 reload 或 back_forward 开始的导航。 • reload 是通过浏览器重新加载操作或 location.

名称	Type	描述	注意
			`reload()` 的导航。 • `back_forward` 是通过浏览器历史遍历操作的导航。 • `prerender` 是由预渲染提示启动的导航。有关更多信息，请参阅预渲染。
startTime	数字	指示触发事件的时间。	值：0 Level 1：entryData.navigationStart - entryData.navigationStart Level 2：entryData.startTime

名称	Type	描述	注意
unloadEventStart	数字	指示引发 unload 事件后，窗口中上一个文档开始卸载的时间。	值：如果不存在上一个文档，或者上一个文档或者其中一个所需的重新导向不是同一来源，则返回值为 0。 Level 1： `entryData.unloadEventStart > 0 ? entryData.unloadEventStart - entryData.navigationStart : 0` Level 2：entryData.unloadEventStart

名称	Type	描述	注意
promptForUnload	数字	卸载文档所用的时间。换言之，unloadEventStart 和 unloadEventEnd 之间的时间。UnloadEventEnd 表示卸载事件处理程序完成的时刻（以毫秒为单位）。	值：如果不存在上一个文档，或者上一个文档或者其中一个所需的重新导向不是同一来源，则返回值为 0。 Level 1：entryData.unloadEventEnd - entryData.unloadEventStart Level 2：entryData.unloadEventEnd - entryData.unloadEventStart

名称	Type	描述	注意
redirectCount	数字	表示在当前浏览的上下文中自上次非重新导向导航以来的重新导向次数。 此属性非必需。	值：如果没有重新导向，或者如果有与目标文档不同源的任何重新导向，则返回值为 0。 Level 1：不可用 Level 2：entryData.redirectCount

名称	Type	描述	注意
redirectStart	数字	第一次 HTTP 重新导向开始的时间。	值：如果没有重新导向，或者如果有与目标文档不同源的任何重新导向，则返回值为 0。 Level 1： `entryData.redirectStart > 0 ? entryData.redirectStart - entryData.navigationStart : 0` Level 2：entryData.redirectStart

名称	Type	描述	注意
redirectTime	数字	HTTP 重新导向所用的时间。这是 `redirectStart` 与 `redirectEnd` 之间的差异。	Level 1：：entryData.redirectEnd - entryData.redirectStart Level 2：：entryData.redirectEnd - entryData.redirectStart

名称	Type	描述	注意
WorkerStart	数字	这是 PerformanceResourceTiming 接口的属性。这标志着工件线程操作开始。 此属性非必需。	值：如果 Service Worker 线程已运行，或紧靠启动 Service Worker 线程之前，则此属性将返回紧靠分派 FetchEvent 前的时间。如果 Service Worker 未拦截资源，则返回 0。 Level 1：不可用 Level 2：entryData.workerStart

名称	Type	描述	注意
workerTime	数字	如果 Service Worker 未拦截资源，则返回工件线程操作所需的时间。 此属性非必需。	Level 1：不可用 Level 2： `entryData.workerStart > 0 ? entryData.fetchStart - entryData.workerStart : 0`
fetchStart	数字	浏览器准备好使用 HTTP 请求获取文档的时间。这在检查任何应用程序缓存之前完成。	Level 1： `: entryData.fetchStart > 0 ? entryData.fetchStart - entryData.navigationStart : 0` Level 2：entryData.fetchStart

名称	Type	描述	注意
domainLookupStar	数字	域查找开始的时间。	值：如果使用持久连接或信息存储在缓存或本地资源中，则该值将与 `fetchStart` 相同。 Level 1： `entryData.domainLookupStart > 0 ? entryData.domainLookupStart - entryData.navigationStart : 0` Level 2：entryData.domainLookupStart

名称	Type	描述	注意
dns	数字	域名查找所需的时间。	值：如果缓存了资源和 DNS 记录，则预期值为 0。 Level 1：entryData.domainLookupEnd - entryData.domainLookupStart Level 2：entryData.domainLookupEnd - entryData.domainLookupStart
nextHopProtocol	String	表示用于获取资源的网络协议的字符串。 此属性非必需。	Level 1：不可用 Level 2：entryData.nextHopProtocol

名称	Type	描述	注意
connectStart	数字	紧靠用户代理开始建立与服务器的连接以检索文档之前的时间。	值：如果使用了 RFC2616 持久连接，或者如果从相关应用程序缓存或本地资源中检索当前文档，则此属性将返回 domainLookupEnd 值。 Level 1： `entryData.connectStart > 0 ? entryData.connectStart - entryData.navigationStart : 0` Level 2：entryData.connectStart

名称	Type	描述	注意
connect	数字	评估建立传输连接或执行 SSL 身份验证所需的时间。其中还包括浏览器发出过多并发请求时所用的受阻时间。	Level 1：entryData.connectEnd - entryData.connectStart Level 2：entryData.connectEnd - entryData.connectStart
secureConnectionStart	数字	如果当前页面的 URL 模式为“https”，则此属性将返回紧靠用户代理启动握手过程以保护当前连接之前的时间。如果未使用 HTTPS，则返回 0。有关 URL 模式的更多信息，请参阅 URL 表示形式。	公式：entryData.secureConnectionStart

名称	Type	描述	注意
tlsTime	数字	完成 SSL 握手所需的时间。	Level 1： `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0` Level 2： `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0`

名称	Type	描述	注意
requestStart	数字	紧靠用户代理开始从服务器、相关应用程序缓存或本地资源请求资源之前的时间。	Level 1： `: entryData.requestStart > 0 ? entryData.requestStart - entryData.navigationStart : 0` Level 2：entryData.requestStart
timeToFirstByte	数字	发出请求后接收信息第一个字节所需的时间。此时间相对于 `startTime` 。	Level 1：entryData.responseStart - entryData.requestStart Level 2：entryData.responseStart - entryData.requestStart

名称	Type	描述	注意
responseStart	数字	紧靠用户代理的 HTTP 解析器从相关应用程序缓存、本地资源或服务器接收响应的第一个字节之后的时间。	Level 1： `entryData.responseStart > 0 ? entryData.responseStart - entryData.navigationStart : 0` Level 2：entryData.responseStart

名称	Type	描述	注意
responseTime	String	以字节形式从相关应用程序缓存、本地资源或服务器接收完整响应所用的时间。	Level 1： `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0` Level 2： `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0`

名称	Type	描述	注意
domInteractive	数字	解析器在主文档上完成工作以及构建 HTML DOM 所用的时间。此时，其 `Document.readyState` 变为“交互式”，并引发相应的 `readystatechange` 事件。	Level 1： `entryData.domInteractive > 0 ? entryData.domInteractive - entryData.navigationStart : 0` Level 2：entryData.domInteractive

名称	Type	描述	注意
domContentLoadedEventStart	数字	表示紧靠用户代理在当前文档中触发 DOMContentLoaded 事件之前的时间值。初始 HTML 文档完全加载和解析后，将触发 DOMContentLoaded 事件。此时，主 HTML 文档已完成解析，浏览器开始构建渲染树，并且还需要加载子资源。此步骤不会等待样式表、图像和子框架完成加载。	Level 1： `entryData.domContentLoadedEventStart > 0 ? entryData.domContentLoadedEventStart - entryData.navigationStart : 0` Level 2：entryData.domContentLoadedEventStart

名称	Type	描述	注意
domContentLoaded	数字	渲染树构建的此开启时间和结束时间使用 `domContentLoadedEventStart` 和 `domContentLoadedEventEnd` 标记。其允许 CloudWatch RUM 跟踪执行情况。此属性是 `domContentLoadedStart` 与 `domContentLoadedEnd` 之间的差异。 在此期间，DOM 和 CSSOM 已准备就绪。此属性将等待脚本执行，但异步和动态创建的脚本除外。如果脚本依赖样式表，`domContentLoaded` 也将等待样式表。其不会等待图像。 **Note** 在 Google Chrome 的网络面板中，`domContentLoadedStart` 和 `domContentLoadedEnd` 的实际值与 `domContentLoaded` 近似。这表示从页面加载过程开始的 HTML DOM + CSSOM 渲染树构建时间。如果是导航指标，`domContentLoaded` 值表示开始和结束值之间的差异，这是仅下载子资源和渲染树构建所需的时间。	Level 2：entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart Level 2：entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart

名称	Type	描述	注意
domComplete	数字	紧靠浏览器将当前文档的当前文档准备就绪状态设置为完成之前的时间。此时，子资源（例如图像）的加载已经完成。这包括下载 CSS 和同步 JavaScript 等阻止内容所用的时间。在 Google Chrome 的网络面板中，此属性与 `loadTime` 近似。	Level 1： `entryData.domComplete > 0 ? entryData.domComplete - entryData.navigationStart : 0` Level 2：entryData.domComplete
domProcessingTime	数字	响应和加载事件开始之间的总时间。	Level 1：entryData.loadEventStart - entryData.responseEnd Level 2：entryData.loadEventStart - entryData.responseEnd

名称	Type	描述	注意
loadEvent Start	数字	紧靠当前文档 `load` 事件触发之前的时间。	Level 1： `entryData.loadEventStart > 0 ? entryData.loadEventStart - entryData.navigationStart : 0` Level 2：entryData.loadEventStart
loadEvent Time	数字	`loadEventStart` 与 `loadEventEnd` 之间的差异。在此期间，将触发等待此加载事件的其他函数或逻辑。	Level 1：entryData.loadEventEnd - entryData.loadEventStart Level 2：entryData.loadEventEnd - entryData.loadEventStart

名称	Type	描述	注意
duration	String	持续时间为页面总加载时间。其记录下载主页面及其所有同步子资源的时间，以及渲染页面的时间。稍后还会继续下载脚本等异步资源。此属性是 `loadEventEnd` 与 `startTime` 属性之间的差异。	Level 1：entryData.loadEventEnd - entryData.navigationStart Level 2：entryData.duration
headerSize	数字	返回 `transferSize` 与 `encodedBodySize` 之间的差异。 此属性非必需。	Level 1：不可用 Level 2：entryData.transferSize - entryData.encodedBodySize Level 2：entryData.transferSize - entryData.encodedBodySize

名称	Type	描述	注意
compressionRatio	数字	`encodedBodySize` 和 `decodedBodySize` 的比率。`encodedBodySize` 的值是不包括 HTTP 标头的资源的压缩大小。`decodedBodySize` 的值是不包括 HTTP 标头的资源的解压大小。 此属性非必需。	Level 1：不可用。 Level 2： `entryData.encodedBodySize > 0 ? entryData.decodedBodySize / entryData.encodedBodySize : 0`
navigationTimingLevel	数字	Navigation timing API 版本。	值：1 或 2

资源事件架构

只有在应用程序监控已激活性能遥测时，才会收集资源事件。

时间戳指标基于 DOMHighResTimeStamp typedef。对于 Level 2 API，默认情况下，所有计时都相对于 `startTime`。但是对于 Level 1 API，需从时间戳指标中减去 `navigationStart` 指标以获取相对值。所有时间戳值均以毫秒为单位。

代理生成的资源事件包含以下属性。

名称	Type	描述	注意
targetUrl	String	返回资源的 URL。	公式：entryData.name

名称	Type	描述	注意
initiatorType	String	表示启动性能资源事件的资源类型。	值："resource" 公式：entryData.initiatorType
duration	String	返回 `responseEnd` 与 `startTime` 属性之间的差异。 此属性非必需。	公式：entryData.duration
transferSize	数字	返回获取的资源大小（八位字节），包括响应标头字段和响应有效负载正文。 此属性非必需。	公式：entryData.transferSize
fileType	String	从目标 URL 模式派生的扩展。	

最大内容绘制事件架构

最大内容绘制事件包含以下属性。

只有在应用程序监控已激活性能遥测时，才会收集这些事件。

名称	描述		
值	有关更多信息，请参阅 Web 重要信息。		

首次输入延迟事件

首次输入延迟事件包含以下属性。

只有在应用程序监控已激活性能遥测时，才会收集这些事件。

名称	描述		
值	有关更多信息，请参阅 Web 重要信息。		

累计布局偏移事件

累计布局转移事件包含以下属性。

只有在应用程序监控已激活性能遥测时，才会收集这些事件。

名称	描述		
值	有关更多信息，请参阅 Web 重要信息。		

HTTP 事件

HTTP 事件包含以下属性。其中将包含 `Response` 字段或 `Error` 字段，但无法二者皆包含。

只有在应用程序监控已激活 HTTP 性能遥测时，才会收集这些事件。

名称	描述
请求	请求字段包含以下内容： • `Method` 字段，其中包含 GET、POST 等值。 • URL
响应	响应字段包含以下内容： • 状态，例如 2xx、4xx 或 5xx • 状态文本
错误	错误字段包含以下内容：

名称	描述
	• 类型 • 消息 • 文件名 • 行号 • 列号 • 堆栈跟踪

X-Ray 跟踪事件架构

只有在应用程序监控已激活 X-Ray 跟踪时，才会收集这些事件。

有关 X-Ray 跟踪事件架构的信息，请参阅 AWS X-Ray 分段文档。

单页应用程序的路由更改计时

在传统的多页面应用程序中，当用户请求加载新内容时，用户实际上是在向服务器请求新的 HTML 页面。因此，CloudWatch RUM Web 客户端使用常规性能 API 指标来捕获加载时间。

但是，单页 Web 应用程序使用 JavaScript 和 Ajax 来更新界面，而无需从服务器加载新页面。浏览器计时 API 不会记录单页更新，而是使用路由更改计时。

CloudWatch RUM 支持监控来自服务器的整页加载和单页更新，但有以下区别：

- 对于路由更改计时，没有浏览器提供的指标，例如 `tlsTime`、`timeToFirstByte` 等等。
- 对于路由更改计时，`initiatorType` 字段将为 `route_change`。

CloudWatch RUM Web 客户端监听可能导致路由变更的用户交互，当记录此类用户交互时，Web 客户端会记录一个时间戳。如果满足以下两个条件，则将开始路由更改计时：

- 浏览器历史记录 API（浏览器前进和后退按钮除外）用于执行路由更改。
- 路由更改检测时间与最新用户交互时间戳之间的差异小于 1000 ms。这样可以避免数据偏斜。

然后，一旦路由更改计时开始，如果没有正在进行的 AJAX 请求和 DOM 更改，则该计时结束。然后，最新完成的活动的时间戳将用作完成时间戳。

如果有持续的 AJAX 请求或 DOM 变更超过 10 秒（默认），则路由更改计时将超时。在这种情况下，CloudWatch RUM Web 客户端将不再记录此路由更改的计时。

因此，路由更改事件的持续时间按以下公式计算：

```
(time of latest completed activity) - (latest user interaction timestamp)
```

管理使用 CloudWatch RUM 的应用程序

使用这些章节中的步骤管理应用程序对 CloudWatch RUM 的使用。

如何查找已生成的代码段？

要查找已为应用程序生成的 CloudWatch RUM 代码段，请按照以下步骤操作。

查找已生成的代码段

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图）。
4. 在应用程序监控的名称旁边，选择 View JavaScript（查看 JavaScript）。
5. 在 JavaScript Snippet（JavaScript 代码段）窗格中，选择 Copy to clipboard（复制到剪贴板）。

编辑应用程序

要更改应用程序监控的设置，请按以下步骤操作。您可以更改除应用程序监控名称之外的任何设置。

编辑应用程序对 CloudWatch RUM 的使用方式

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图）。
4. 选择应用程序名称旁边的按钮，然后选择 Actions（操作）、Edit（编辑）。
5. 更改除应用程序监控名称之外的任何设置。有关设置的更多信息，请参阅 步骤 2：创建应用程序监控。
6. 完成后，选择保存。

更改设置会更改代码段。现在，您必须将更新后的代码段粘贴到应用程序中。

7. 在创建 JavaScript 代码段之后，选择 Copy to clipboard（复制到剪贴板）或 Download（下载），然后选择 Done（完成）。

   要使用新设置开启监控，请将代码段插入应用程序中。在 <body> 元素或任何其他 <script> 标签之前，将代码段插入到应用程序的 <head> 元素中。

停止使用 CloudWatch RUM 或删除应用程序监控

要停止在应用程序中使用 CloudWatch RUM，请从应用程序代码中删除 RUM 生成的代码段。

要删除 RUM 应用程序监控，请按以下步骤操作。

删除应用程序监控

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、RUM。
3. 选择 List view（列表视图）。
4. 选择应用程序名称旁边的按钮，然后选择 Actions（操作）、Delete（删除）。
5. 在确认框中，输入 **Delete**，然后选择 Delete（删除）。
6. 如果您尚未执行此操作，请从应用程序代码中删除 CloudWatch RUM 代码段。

CloudWatch RUM 配额

CloudWatch RUM 具有以下配额。

资源	默认限额
应用程序监控	每个账户 20 个 您可以请求提高限额。
RUM 摄取率	每秒 50 个 PutRumEvents 请求（TPS）。 您可以请求提高限额。

CloudWatch RUM 故障排除

本章节包含的提示可以帮助您对 CloudWatch RUM 进行故障排除。

应用程序没有数据

首先，请确保代码段已正确插入应用程序中。有关更多信息，请参阅 步骤 4：将代码段插入应用程序。

如果这不是问题所在，则可能是没有流量传入应用程序。通过与用户相同的方式访问应用程序来生成一些流量。

已停止记录应用程序的数据

应用程序可能经更新，现在不再包含 CloudWatch RUM 代码段。检查应用程序代码。

另一种可能性是，有人可能已经更新了代码段，但后来没有将更新的代码段插入应用程序中。按照 如何查找已生成的代码段？ 中的说明查找当前正确的代码段，然后将其与粘贴到应用程序中的代码段进行比较。

使用 CloudWatch Evidently 执行启动和 A/B 实验

通过在推出新功能时向指定百分比的用户提供新功能，您可以使用 Amazon CloudWatch Evidently 对功能进行安全地验证。您可以监控新功能的性能，以帮助您决定何时向用户增加流量。这有助于您在完全启动该功能之前，降低风险并识别意外后果。

您还可以进行 A/B 实验，以根据证据和数据制定功能设计决策。一个实验可以同时测试多达五个功能变体。Evidently 会收集实验数据并使用统计方法对其进行分析。它还会就哪些功能变体的性能更好提供明确建议。您可以测试面向用户的功能和后端功能。

Evidently 定价

Evidently 会根据 Evidently 事件和 Evidently 分析单位向您的账户收取费用。Evidently 事件包括单击量和页面浏览量等数据事件以及确定向用户提供的功能变体的任务事件。

Evidently 分析单元根据您在 Evidently 中创建的规则，从 Evidently 事件中生成。分析单元是事件的规则匹配数。例如，用户单击事件可能会产生单个 Evidently 分析单元，即单击计数。另一个示例是用户结算事件，其可能会产生两个 Evidently 分析单元，即结算值和购物车中的项目数量。有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

CloudWatch Evidently 目前在以下区域中可用：

- 美国东部（俄亥俄）
- 美国东部（弗吉尼亚州北部）
- 美国西部（俄勒冈州）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 欧洲地区（法兰克福）
- 欧洲（爱尔兰）
- 欧洲地区（斯德哥尔摩）

主题

- 使用 Evidently 的 IAM policy
- 创建项目、功能、启动和实验
- 管理功能、启动和实验
- 将代码添加到应用程序
- 项目数据存储
- Evidently 是如何计算结果的
- 在控制面板中查看启动结果
- 在控制面板中查看实验结果
- CloudWatch Evidently 如何收集和存储数据
- 使用 Evidently 的服务相关角色
- CloudWatch Evidently 配额
- 教程：使用 Evidently 示例应用程序进行 A/B 测试

使用 Evidently 的 IAM policy

要完全管理 CloudWatch Evidently，您必须以具有以下权限的 IAM 用户或角色的身份登录：

- AmazonCloudWatchEvidentlyFullAccess 策略
- ResourceGroupsandTagEditorReadOnlyAccess 策略

此外，为能够创建在 Amazon S3 或 CloudWatch Logs 中存储评估事件的项目，您需要以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:DescribeResourcePolicies",
                "logs:PutResourcePolicy"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

CloudWatch RUM 集成的其他权限

此外，如果您打算管理与 Amazon CloudWatch RUM 集成的 Evidently 启动或实验，并使用 CloudWatch RUM 指标进行监控，您需要 AmazonCloudWatchRUMFullAccess 策略。要创建 IAM 角色以授予 CloudWatch RUM Web 客户端向 CloudWatch RUM 发送数据的权限，您需要以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
```

```
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*",
                "arn:aws:iam::*:policy/service-role/CloudWatchRUMEvidentlyPolicy-*"
            ]
        }
    ]
}
```

对 Evidently 的只读访问权限

对于需要查看 Evidently 数据但不需要创建 Evidently 资源的其他用户，您可以授予 AmazonCloudWatchEvidentlyReadOnlyAccess 策略。

创建项目、功能、启动和实验

无论是出于功能启动或者 A/B 实验目的，若要开始使用 CloudWatch Evidently，您都要先创建项目。项目是资源的逻辑分组。在项目中，您可以创建具有要测试或启动的变体的功能。您可以在创建启动或实验之前创建功能，也可以同时创建。

主题

- 创建新 项目
- 使用客户端评估 - 由 AWS AppConfig 提供支持
- 向项目添加功能
- 使用细分来聚焦受众
- 创建启动
- 创建实验

创建新 项目

请使用以下步骤设置新的 CloudWatch Evidently 项目。

要创建新的 CloudWatch Evidently 项目

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择 Create project（创建项目）。

4. 对于 Project name（项目名称），输入一个名称，用于在 CloudWatch Evidently 控制台中标识此项目。

   您可以选择添加项目描述。

5. 对于 Evaluation event storage（评估事件存储），请选择是否要存储使用 Evidently 收集的评估事件。即使您没有存储这些事件，Evidently 也会将其聚合以创建可在 Evidently 控制面板中查看的指标和其他实验数据。有关更多信息，请参阅 项目数据存储。

6. 对于 Use client-side evaluation（使用客户端评估），请选择是否要为此项目启用客户端评估。通过客户端评估，您的应用程序可以在本地为用户会话分配变体，而不是调用 EvaluateFeature 操作来分配。这样可以减少 API 调用带来的延迟和可用性风险。有关更多信息，请参阅 使用客户端评估 - 由 AWS AppConfig 提供支持。

   要创建具有客户端评估的项目，您必须具有 `evidently:ExportProjectAsConfiguration` 权限。

   如果您启用客户端评估，还要执行以下操作：

   a. 选择是使用现有 AWS AppConfig 应用程序还是创建一个新的应用程序。
   b. 选择是使用现有 AWS AppConfig 环境还是创建一个新的环境。

      有关 AWS AppConfig 中的应用程序和环境的更多信息，请参阅 AWS AppConfig 的工作原理。

7. （可选）要向此项目添加标签，请选择 Tags（标签）、Add new tag（添加新标签）。

   然后，对于 Key（键），输入标签的名称。您可以在 Value (值) 中添加可选的标签值。

   要添加其他标签，请再次选择 Add new tag（添加新标签）。

   有关更多信息，请参阅标记 AWS 资源。

8. 请选择 Create project（创建项目）。

使用客户端评估 - 由 AWS AppConfig 提供支持

您可以在项目中使用客户端评估 - 由 AWS AppConfig 提供支持（客户端评估），以使您的应用程序能够在本地为用户会话分配变体，而不是通过调用 EvaluateFeature 操作分配变体。这样可以减少 API 调用带来的延迟和可用性风险。

要使用客户端评估，请附上 AWS AppConfig Lambda 扩展程序作为您的 Lambda 函数的一个层并配置环境变量。客户端评估在本地主机上作为侧进程运行。然后，您可以针对 `localhost` 调用

EvaluationFeature 和 PutProjectEvent。客户端评估过程负责变体分配、缓存和数据同步。有关 AWS AppConfig 的更多信息，请参阅 AWS AppConfig 的工作原理。

在与 AWS AppConfig 集成时，可以向 Evidently 指定一个 AWS AppConfig 应用程序 ID 和一个 AWS AppConfig 环境 ID。您可以在各个 Evidently 项目之间使用相同的应用程序 ID 和环境 ID。

在您创建一个项目时，如果该项目启用了客户端评估，Evidently 会为该项目创建一个 AWS AppConfig 配置文件。每个项目的配置文件将有所不同。

客户端评估访问控制

Evidently 客户端评估使用的访问控制机制与 Evidently 的其余部分不同。我们强烈建议您了解这一点，以便实施适当的安全措施。

利用 Evidently，您可以创建 IAM policy 来限制用户可对各个资源执行的操作。例如，您可以创建不允许用户执行 EvaluateFeature 操作的用户角色。有关可通过 IAM policy 控制的 Evidently 操作的更多信息，请参阅 Amazon CloudWatch Evidently 定义的操作。

可通过客户端评估模型对使用项目元数据的 Evidently 功能进行本地评估。启用了客户端评估的项目的用户可对本地主机端点调用 EvaluateFeature API，此 API 调用不会到达 Evidently，也不会通过 Evidently 服务的 IAM policy 进行身份验证。即使用户没有相应 IAM 权限来使用 EvaluateFeature 操作，此调用也会成功。但是，用户仍然需要 PutProjectEvents 权限使代理缓冲评估事件或自定义事件，以及将数据异步分流到 Evidently。

此外，用户必须具有 `evidently:ExportProjectAsConfiguration` 权限才能创建使用客户端评估的项目。这样有助于您控制对客户端评估期间调用的 EvaluateFeature 操作的访问权限。

如果您不注意，客户端评估安全模型可能会推翻您对 Evidently 其余部分设置的策略。具有 `evidently:ExportProjectAsConfiguration` 权限的用户可以创建启用了客户端评估的项目，然后使用 EvaluateFeature 操作对该项目进行客户端评估，即使他们在 IAM policy 中被明确拒绝执行 EvaluateFeature 操作。

Lambda 入门

Evidently 当前通过使用 AWS Lambda 环境支持客户端评估。首先，请确定要使用哪一个 AWS AppConfig 应用程序和环境。选择现有的应用程序和环境，或创建新的应用程序和环境。

以下示例 AWS AppConfig AWS CLI 命令可创建应用程序和环境。

```
aws appconfig create-application --name YOUR_APP_NAME
```

```
aws appconfig create-environment --application-id YOUR_APP_ID --
name YOUR_ENVIRONMENT_NAME
```

接下来，使用这些 AWS AppConfig 资源创建一个 Evidently 项目。有关更多信息，请参阅 创建新 项目。

可在 Lambda 中使用 Lambda 层支持客户端评估。该层是一个公共层，属于 AWS-AppConfig-Extension（AWS AppConfig 服务创建的一个公共 AWS AppConfig 扩展程序）的一部分。有关 Lambda 层的更多信息，请参阅层。

要使用客户端评估，您必须将此层添加到 Lambda 函数并配置权限和环境变量。

要将 Evidently 客户端评估 Lambda 层添加到您的 Lambda 函数并进行配置

1. 如果尚未创建 Lambda 函数，请创建一个。
2. 将客户端评估层添加到您的函数中。您可以指定它的 ARN，也可以从 AWS 层的列表中进行选择（如果还没有此 ARN）。有关更多信息，请参阅配置函数以使用层和 AWS AppConfig Lambda 扩展程序的可用版本。
3. 创建名为 EvidentlyAppConfigCachingAgentPolicy 且具有以下内容的 IAM policy，并将其附加到此函数的执行角色。有关更多信息，请参阅 Lambda 执行角色。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "VisualEditor0",
               "Effect": "Allow",
               "Action": [
                   "appconfig:GetLatestConfiguration",
                   "appconfig:StartConfigurationSession",
                   "evidently:PutProjectEvents"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

4. 向您的 Lambda 函数添加所需的环境变量 AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS。此环境变量用于指定 Evidently 项目与 AWS AppConfig 资源之间的映射。

如果您将此函数用于一个 Evidently 项目，请将环境变量的值设置为：applications/*APP_ID*/environments/*ENVIRONMENT_ID*/configurations/*PROJECT_NAME*

如果您将此函数用于多个 Evidently 项目，请使用逗号分隔各个值，如以下示例所示：applications/*APP_ID_1*/environments/*ENVIRONMENT_ID_1*/configurations/*PROJECT_NAME_1*, applications/*APP_ID_2*/environments/*ENVIRONMENT_ID_2*/configurations/*PROJECT_NAME_2*

5. （可选）设置其他环境变量。有关更多信息，请参阅配置 AWS AppConfig Lambda 扩展程序。
6. 在您的应用程序中，通过将 EvaluateFeature 发送到 localhost 进行本地 Evidently 评估。

Python 示例：

```
import boto3
from botocore.config import Config

def lambda_handler(event, context):
    local_client = boto3.client(
        'evidently',
        endpoint_url="http://localhost:2772",
        config=Config(inject_host_prefix=False)
    )
    response = local_client.evaluate_feature(
        project=event['project'],
        feature=event['feature'],
        entityId=event['entityId']
    )
    print(response)
```

Node.js 示例：

```
const AWS = require('aws-sdk');
const evidently = new AWS.Evidently({
    region: "us-west-2",
    endpoint: "http://localhost:2772",
    hostPrefixEnabled: false
});

exports.handler = async (event) => {

    const evaluation = await evidently.evaluateFeature({
```

```
        project: 'John_ETCProject_Aug2022',
        feature: 'Feature_IceCreamFlavors',
        entityId: 'John'
    }).promise()

    console.log(evaluation)
    const response = {
        statusCode: 200,
        body: evaluation,
    };
    return response;
};
```

Kotlin 示例：

```
String localhostEndpoint = "http://localhost:2772/"
public AmazonCloudWatchEvidentlyClient getEvidentlyLocalClient() {
    return AmazonCloudWatchEvidentlyClientBuilder.standard()

 .withEndpointConfiguration(AwsClientBuilder.EndpointConfiguration(localhostEndpoint,
 region))

 .withClientConfiguration(ClientConfiguration().withDisableHostPrefixInjection(true))
        .withCredentials(credentialsProvider)
        .build();
}

AmazonCloudWatchEvidentlyClient evidently = getEvidentlyLocalClient();

// EvaluateFeature via local client.
EvaluateFeatureRequest evaluateFeatureRequest = new
 EvaluateFeatureRequest().builder()
   .withProject(${YOUR_PROJECT}) //Required.
   .withFeature(${YOUR_FEATURE}) //Required.
   .withEntityId(${YOUR_ENTITY_ID}) //Required.
   .withEvaluationContext(${YOUR_EVAL_CONTEXT}) //Optional: a JSON object of
 attributes that you can optionally pass in as part of the evaluation event sent to
 Evidently.
   .build();

EvaluateFeatureResponse evaluateFeatureResponse =
 evidently.evaluateFeature(evaluateFeatureRequest);
```

```
// PutProjectEvents via local client.
PutProjectEventsRequest putProjectEventsRequest = new
 PutProjectEventsRequest().builder()
    .withData(${YOUR_DATA})
    .withTimeStamp(${YOUR_TIMESTAMP})
    .withType(${YOUR_TYPE})
    .build();

PutProjectEvents putProjectEventsResponse =
 evidently.putProjectEvents(putProjectEventsRequest);
```

配置客户端向 Evidently 发送数据的频率

要指定客户端评估向 Evidently 发送数据的频率，您可以选择配置两个环境变量。

- AWS_APPCONFIG_EXTENSION_EVIDENTLY_EVENT_BATCH_SIZE 用于指定每个项目中在将事件发送到 Evidently 之前要批处理的事件数量。有效范围是介于 1 与 50 之间的整数，默认值为 40。
- AWS_APPCONFIG_EXTENSION_EVIDENTLY_BATCH_COLLECTION_DURATION 用于指定在将事件发送到 Evidently 之前等待事件的持续时间（以秒为单位）。默认值为 30。

故障排除

使用以下信息有助于排查在对 CloudWatch Evidently 使用由 AWS AppConfig 提供支持的客户端评估时发生的问题。

调用 EvaluateFeature 操作时出现错误（BadRequestException）：提供的路径不支持 HTTP 方法

您的环境变量可能配置不正确。例如，您可能将 EVIDENTLY_CONFIGURATIONS 用作环境变量名称，而不是 AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS。

ResourceNotFoundException：未找到部署

您对项目元数据的更新尚未部署到 AWS AppConfig。在您用于客户端评估的 AWS AppConfig 环境中检查是否存在活动部署。

ValidationException：没有对项目进行 Evidently 配置

您的 AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS 环境变量可能使用不正确的项目名称进行了配置。

向项目添加功能

CloudWatch Evidently 中的功能代表您要启动或要测试其变体的功能。

必须先创建项目，才能添加功能。有关更多信息，请参阅 创建新 项目。

向项目添加功能

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择项目名称。
4. 请选择 Add feature（添加功能）。
5. 对于 Feature name（功能名称），输入用于标识此项目中此功能的名称。

   您也可以添加功能描述。
6. 对于 Feature variations（功能变体）和 Variation type（变体类型），请选择 Boolean、Long（长整数）、Double（双精度）或者 String（字符串）。有关更多信息，请参阅 变体类型。
7. 最多可向功能添加五个变体。每个变体的 Value（值）必须对您选择的 Variation type（变体类型）有效。

   指定其中一个变体作为默认变体。这是将与其他变体进行比较的基准，并且应作为现在向用户提供的变体。这也是向未添加到此功能的启动或实验中的用户提供的变体。
8. 请选择 Sample code（示例代码）。此代码示例显示了为设置变体并为其分配用户会话，您需要添加到应用程序中的内容。您可以在 JavaScript、Java 和 Python 之间选择代码。

   您现在无需将代码添加到应用程序中，但必须在开始启动或实验之前添加。

   有关更多信息，请参阅 将代码添加到应用程序。
9. （可选）要指定某些用户可以始终看到某个变体，请选择 Overrides（覆盖）、Add override（添加覆盖）。然后，通过在 Identifier（标识符）中输入用户 ID、账户 ID 或其他一些标识符，指定用户并指定其可以看到的变体。

   如果您想确保自己的测试团队成员或其他内部用户可以看到特定变体，这非常有用。已分配覆盖的用户的会话无益于启动或试验指标。

   您可以通过再次选择添加覆盖，为多达 20 个用户重复此操作。
10. （可选）要向此功能添加标签，请选择 Tags（标签）、Add new tag（添加新标签）。

然后，对于 Key（键），输入标签的名称。您可以在 Value (值) 中添加可选的标签值。

要添加其他标签，请再次选择 Add new tag（添加新标签）。

有关更多信息，请参阅标记 AWS 资源。

11. 请选择 Add feature（添加功能）。

变体类型

创建功能并定义变体时，必须选择 variation type（变体类型）。可能的类型如下：

- 布尔值
- 长整数
- 双精度浮点数
- String

变体类型设置了代码中不同变体的区分方式。您可以使用变体类型来简化 CloudWatch Evidently 的实施，也可以简化启动和实验中的功能修改过程。

例如，如果您使用长整数变体类型定义功能，则指定用于区分变体的整数可以是直接传递到代码中的数字。示例可能为按钮的像素大小测试。变体类型的值可以是每个变体中所使用的像素数。每个变体的代码可以读取变体类型值并将其用作按钮大小。要测试新的按钮大小，您可以更改用于变体值的数字，而无需更改任何其他代码。

除想要进行 A/A 测试以初步试用 CloudWatch Evidently 或者出于其他目的这么做之外，当您在功能中为变体类型设置值时，应避免向多个变体分配相同的值。

当将 JSON 作为类型时，Evidently 不向其提供本机支持，但您可以在 String（字符串）变体类型中传入 JSON，并在代码中解析该 JSON。

使用细分来聚焦受众

您可以定义受众细分并将它们用于发布和实验中。细分是您的受众中具有一个或多个相同特征的部分。例如 Chrome 浏览器用户、欧洲用户或欧洲的 Firefox 浏览器用户，他们也符合您的应用程序收集的其他标准，例如年龄。

在实验中使用细分将该实验限制为仅评估符合细分标准的用户。当您在发布中使用一个或多个细分时，您可以为不同的受众细分定义不同的流量拆分。

细分规则模式语法

要创建细分，请定义细分规则模式。指定要用于评估用户会话是否在细分中的属性。将您创建的模式与 Evidently 在用户会话中找到的 evaluationContext 的值进行比较。有关更多信息，请参阅 使用 EvaluateFeature。

要创建细分规则模式，请指定希望模式匹配的字段。您也可以在模式中使用逻辑，例如 And、Or、Not 和 Exists。

为了使 evaluationContext 匹配模式，evaluationContext 必须匹配规则模式的所有部分。Evidently 忽略了 evaluationContext 中并不包含在规则模式中的字段。

规则模式匹配的值遵循 JSON 规则。可以包括用引号（"）括起来的字符串、数字和关键字 true、false 和 null。

对于字符串，Evidently 使用精确的逐个字符匹配，而不进行小写化或任何其他字符串标准化。因此，规则匹配区分大小写。例如，如果您的 evaluationContext 包含一个 browser 属性，但您的规则模式检查 Browser，则它将不匹配。

此外，Evidently 使用字符串表示。例如，300、300.0 和 3.0e2 不相等。

在编写规则模式来匹配 evaluationContext 时，您可以使用 TestSegmentPattern API 或 test-segment-pattern CLI 命令以测试模式是否匹配正确的 JSON。有关更多信息，请参阅 TestSegmentPattern。

以下摘要显示了可在 Evidently 细分模式使用的所有比较运算符。

Comparison（比较）	示例	Rule syntax（规则语法）
Null	用户 ID 为空	`{ "UserID": [ null ] }`
空	姓氏为空	`{ "LastName": [""] }`
等于	浏览器是“Chrome”	`{`

Comparison（比较）	示例	Rule syntax（规则语法）
		"Browser": ["Chrome"] }
并且	国家/地区为“"法国”，设备为“移动设备”	{ "Country": ["France"], "Device": ["Mobile"] }
或（单个属性的多个值）	浏览器是“Chrome”或“Firefox”	{ "Browser": ["Chrome", "Firefox"] }
或（不同的属性）	浏览器为“Safari”或设备为“平板电脑”	{ "$or": [{"Browser": ["Safari"]}, {"Device": ["Tablet" }]] }
Not	浏览器是除“Safari”以外的任何浏览器	{ "Browser": [{ "anything-but": ["Safari"] }] }
数值（等于）	价格为 100	{ "Price": [{ "numeric": ["=", 100] }] }

Comparison（比较）	示例	Rule syntax（规则语法）
数值（范围）	价格大于 10，且小于等于 20	`{ "Price": [ { "numeric": [ ">", 10, "<=", 20 ] } ] }`
存在	年龄字段存在	`{ "Age": [ { "exists": true } ] }`
不存在	年龄字段不存在	`{ "Age": [ { "exists": false } ] }`
开头为前缀	地区位于美国	`{ "Region": [ {"prefix": "us-" } ] }`
结尾为后缀	位置有后缀“West”	`{ "Region": [ {"suffix": "West" } ] }`

细分规则示例

以下所有示例都假定您正在使用您在规则模式中使用的相同字段标签和值传递 `evaluationContext` 的值。

如果 `Browser` 为 Chrome 或 Firefox 且 `Location` 为美国西部，则以下示例匹配。

```
{
```

```
  "Browser": ["Chrome", "Firefox"],
  "Location": ["US-West"]
}
```

如果 Browser 是除 Chrome 以外的浏览器，Location 以 US 开头且 Age 字段存在，则以下示例匹配。

```
{
  "Browser": [ {"anything-but": ["Chrome"]}],
  "Location": [{"prefix": "US"}],
  "Age": [{"exists": true}]
}
```

如果 Location 为日本且 Browser 为 Safari 或 Device 为平板设备，则以下示例匹配。

```
{
 "Location": ["Japan"],
 "$or": [
   {"Browser": ["Safari"]},
   {"Device": ["Tablet"]}
 ]
}
```

创建分段

创建细分后，您即可在任何项目的任何发布或实验中使用它。

创建分段

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 选择 Segments（细分）选项卡。
4. 选择创建客户细分。
5. 对于 Segment name（细分名称），请输入用于标识此细分的名称。

   或者，添加描述。
6. 对于 Segment pattern（细分模式），请输入定义规则模式的 JSON 块。有关规则模式语法的更多信息，请参阅 细分规则模式语法。

创建启动

要向指定百分比的用户公开新功能或更改，请创建启动。然后，在向所有用户推出该功能之前，您可以监控关键指标，例如页面加载时间和转化率。

必须先创建项目，然后才能添加启动。有关更多信息，请参阅 创建新 项目。

添加启动时，您可以使用已创建的功能，也可以在创建启动时创建新功能。

向项目添加启动

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 选择项目名称旁边的按钮，然后选择 Project actions（项目操作）、Create launch（创建启动）。
4. 对于 Launch name（启动名称），输入用于标识此项目中此功能的名称。

   您也可以添加描述。
5. 选择 Select from existing features（从现有功能中选择）或者 Add new feature（添加新功能）。

   如果您使用的是现有功能，请在 Feature name（功能名称）下进行选择。

   如果您选择 Add new feature（添加新功能），请执行以下操作：

   a. 对于 Feature name（功能名称），输入用于标识此项目中此功能的名称。您也可以添加描述。

   b. 对于 Feature variations（功能变体）和 Variation type（变体类型），请选择 Boolean、Long（长整数）、Double（双精度）或者 String（字符串）。有关更多信息，请参阅 变体类型。

   c. 最多可向功能添加五个变体。每个变体的 Value（值）必须对您选择的 Variation type（变体类型）有效。

      指定其中一个变体作为默认变体。这是将与其他变体进行比较的基准，并且应作为现在向用户提供的变体。如果您停止实验，将向所有用户提供此默认变体。

   d. 请选择 Sample code（示例代码）。此代码示例显示了为设置变体并为其分配用户会话，您需要添加到应用程序中的内容。您可以在 JavaScript、Java 和 Python 之间选择代码。

      您现在无需将代码添加到应用程序中，但必须在开始启动之前添加。

有关更多信息，请参阅 将代码添加到应用程序。

6. 对于 Launch configuration（启动配置），请选择立即开始启动或者计划稍后开始启动。
7. （可选）要为已定义的受众细分指定不同的流量拆分，而不是将用于一般受众的流量拆分，请选择 Add Segment Overrides（添加细分覆盖）。

   在 Segment Overrides（细分覆盖）中，选择一个细分并定义要用于该细分的流量拆分。

   您可以选择定义更多细分来定义流量拆分，方法是选择 Add Segment Override（添加细分覆盖）。一次发布最多可以有六个细分覆盖。

   有关更多信息，请参阅 使用细分来聚焦受众。
8. 对于 Traffic configuration（流量配置），选择要分配给不匹配细分覆盖的普通受众的每个变化的流量百分比。您也可以选择不向用户提供变体。

   Traffic summary（流量摘要）显示了可用于本次启动的总流量。
9. 如果您选择计划稍后开始启动，则可以为启动添加多个步骤。每个步骤都可以使用不同的百分比来提供变体。为此，请选择 Add another step（添加其他步骤），然后指定下一步的时间表和流量百分比。您可在启动中包含最多五个步骤。
10. 如果您想在启动期间使用指标跟踪功能性能，请依次选择 Metrics（指标）、Add metric（添加指标）。您可以使用 CloudWatch RUM 指标或自定义指标。

    要使用自定义指标，您可以使用 Amazon EventBridge 规则在此处创建指标。要创建自定义指标，请执行以下操作：

    - 选择 Custom metrics（自定义指标）并输入指标名称。
    - 对于 Entity ID（实体 ID），请在 Metric rule（指标规则）下，输入标识实体的方法。这可以是执行会导致记录指标值的操作的用户或会话。例如，`userDetails.userID`。
    - 对于 Value key（值键），请输入要跟踪以生成指标的值。
    - 您也可以选择输入指标的单位名称。这个用于 Evidently 控制台中图表的单位名称仅供显示之用。

    输入这些字段时，此框会显示如何对 EventBridge 规则进行编码以创建指标的示例。有关 EventBridge 的更多信息，请参阅什么是 Amazon EventBridge？

    要使用 RUM 指标，您必须为应用程序设置 RUM 应用程序监控。有关更多信息，请参阅 设置应用程序以使用 CloudWatch RUM。

> **Note**
>
> 如果您使用 RUM 指标，并且应用程序监控未配置为对 100% 的用户会话进行采样，则并非所有参与启动的用户会话都会将指标发送给 Evidently。为确保启动指标准确无误，我们建议应用程序监控使用 100% 的用户会话进行采样。

11. （可选）如果您为启动创建了至少一个指标，便可将现有 CloudWatch 告警与此启动相关联。为此，请选择 Associate CloudWatch alarms（关联 CloudWatch 告警）。

    当您将告警与启动相关联时，CloudWatch Evidently 必须向告警添加带有项目名称和启动名称的标签。这样一来，CloudWatch Evidently 便可以在控制台的启动信息中显示正确的告警。

    要确认 CloudWatch Evidently 会添加这些标签，请选择 Allow Evidently to tag the alarm resource identified below with this launch resource.（允许 Evidently 用此启动资源标记以下标识的告警资源。）然后，选择 Associate alarm（关联告警），输入告警名称。

    有关创建 CloudWatch 告警的信息，请参阅 使用 Amazon CloudWatch 告警。
12. （可选）要向此启动添加标签，请选择 Tags（标签）、Add new tag（添加新标签）。

    然后，对于 Key（键），输入标签的名称。您可以在 Value (值) 中添加可选的标签值。

    要添加其他标签，请再次选择 Add new tag（添加新标签）。

    有关更多信息，请参阅标记 AWS 资源。
13. 选择 Create launch（创建启动）。

创建实验

使用实验来测试功能或网站的不同版本，并从真实用户会话中收集数据。这样，您便可以根据证据和数据为应用程序做出选择。

在添加实验之前，您必须创建项目。有关更多信息，请参阅 创建新 项目。

添加实验时，您可以使用已创建的功能，或在创建实验时创建新功能。

向项目添加实验

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。

3. 选择项目名称旁边的按钮，然后选择 Project actions（项目操作）、Create experiment（创建实验）。
4. 对于 Experiment name（实验名称），输入用于标识此项目中此功能的名称。

   您也可以添加描述。
5. 选择 Select from existing features（从现有功能中选择）或者 Add new feature（添加新功能）。

   如果您使用的是现有功能，请在 Feature name（功能名称）下进行选择。

   如果您选择 Add new feature（添加新功能），请执行以下操作：

   a. 对于 Feature name（功能名称），输入用于标识此项目中此功能的名称。您也可以选择输入描述。
   b. 对于 Feature variations（功能变体）和 Variation type（变体类型），请选择 Boolean、Long（长整数）、Double（双精度）或者 String（字符串）。该类型定义了每个变体使用哪种类型的值。有关更多信息，请参阅 变体类型。
   c. 最多可向功能添加五个变体。每个变体的 Value（值）必须对您选择的 Variation type（变体类型）有效。

      指定其中一个变体作为默认变体。这是将与其他变体进行比较的基准，并且应作为现在向用户提供的变体。如果您停止使用此功能的实验，则将向之前参加实验的用户百分比提供该默认变体。
   d. 请选择 Sample code（示例代码）。此代码示例显示了为设置变体并为其分配用户会话，您需要添加到应用程序中的内容。您可以在 JavaScript、Java 和 Python 之间选择代码。

      您现在无需将代码添加到应用程序中，但必须在开始实验之前添加。有关更多信息，请参阅 将代码添加到应用程序。
6. 对于 Audience（受众），如果您希望此实验仅应用于与该细分匹配的用户，则可以选择您已创建的细分。有关细分的更多信息，请参阅 使用细分来聚焦受众。
7. 对于 Traffic split for the experiment（实验的流量拆分），指定将在实验中使用会话的选定受众的百分比。然后为实验使用的不同变体分配流量。

   如果同一功能同时运行启动和实验，会将受众首先引至启动。然后，总体受众会产生为启动指定的流量百分比。您在此处指定的百分比是用于实验的剩余受众的百分比。将向之后的任何剩余流量提供默认变体。
8. 对于 Metrics（指标），请选择用于评估实验期间变体的指标。必须使用至少一个指标进行评估。

   a. 对于 Metric source（指标来源），请选择使用 CloudWatch RUM 指标或自定义指标。

b. 输入指标的名称。对于 Goal（目标），如果您希望表示变体更好的指标值较高，请选择 Increase（增加）。如果您希望表示变体更好的指标值较低，请选择 Decrease（减少）。

c. 如果您正在使用自定义指标，可以使用 Amazon EventBridge 规则在此创建指标。要创建自定义指标，请执行以下操作：

- 对于 Entity ID（实体 ID），请在 Metric rule（指标规则）下输入标识实体的方法，这可以是执行导致记录指标值的操作的用户或会话。例如，`userDetails.userID`。
- 对于 Value key（值键），请输入要跟踪以生成指标的值。
- 您也可以选择输入指标的单位名称。这个用于 Evidently 控制台中图表的单位名称仅供显示之用。

仅当已设置 RUM 以监控此应用程序时，才能使用 RUM 指标。有关更多信息，请参阅 CloudWatch RUM。

> **Note**
>
> 如果您使用 RUM 指标，并且应用程序监控未配置为对 100% 的用户会话进行采样，则并非实验中的所有用户会话都会将指标发送给 Evidently。为确保实验指标准确无误，我们建议应用程序监控使用 100% 的用户会话进行采样。

d. （可选）要添加更多要评估的指标，请选择 Add metric（添加指标）。在实验期间，您可以评估多达三个指标。

9. （可选）要创建用于此实验的 CloudWatch 告警，请选择 CloudWatch alarms（CloudWatch 告警）。告警可以监控每个变体与默认变体之间的结果差异是否大于您指定的阈值。如果变体的性能比默认变体差，且差异大于阈值，便会进入告警状态并通知您。

在此处创建告警意味着为每个非默认变体的变体创建告警。

如果您创建告警，请指定以下内容：

- 对于 Metric name（指标名称），请选择要用于告警的实验指标。
- 对于 Alarm condition（告警状态），当变体指标值与默认变体指标值进行比较时，请选择引发告警进入告警状态的条件。例如，如果数字较大表示变体的性能不佳，请选择 Greater（更大）或 Greater/Equal（更大/等于）。例如，如果指标用于测量页面加载时间，此选项会非常合适。
- 输入阈值数字，这是会引发告警进入 `ALARM` 状态的性能差异百分比。
- 对于 Average over period（期间内的平均值），选择比较之前每个变体的指标数据聚合数量。

您可以再次选择 Add new alarm（添加新告警），以向实验添加更多告警。

下一步，选择 Set notifications for the alarm（设置告警通知），然后选择或创建用于向其发送告警通知的 Amazon Simple Notification Service 主题。有关更多信息，请参阅 设置 Amazon SNS 通知。

10. （可选）要向此实验添加标签，请选择 Tags（标签）、Add new tag（添加新标签）。

    然后，对于 Key（键），输入标签的名称。您可以在 Value (值) 中添加可选的标签值。

    要添加其他标签，请再次选择 Add new tag（添加新标签）。

    有关更多信息，请参阅标记 AWS 资源。
11. 选择 Create experiment (创建实验)。
12. 如果您尚未执行，请将功能变体构建到应用程序中。
13. 选择完成。只有您将其启动，实验才会开始。

完成以下过程中的步骤后，实验会立即开始。

开始进行已创建的实验

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择项目名称。
4. 请选择 Experiments（实验）选项卡。
5. 选择实验名称旁边的按钮，然后依次选择 Actions（操作）、Start experiment（开始实验）。
6. （可选）要查看或修改创建实验设置时指定的实验设置，请选择 Experiment setup（实验设置）。
7. 请选择实验结束的时间。
8. 请选择 Start experiment（开始实验）。

   实验会立即开始。

管理功能、启动和实验

使用这些部分中的步骤来管理您已创建的功能、启动和实验。

主题

- 查看功能的当前评估规则和受众流量
- 修改启动流量
- 修改启动的未来步骤
- 修改实验流量
- 停止启动
- 停止实验

查看功能的当前评估规则和受众流量

您可以使用 CloudWatch Evidently 控制台，了解该功能的评估规则如何在功能的当前启动、实验和变体之间分配受众流量。

查看功能的受众流量

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 选择包含该功能的项目名称。
4. 选择 Features（功能）选项卡。
5. 选择功能的名称。

   在 Evaluation rules（评估规则）选项卡中，您可以查看功能的受众流量流动，如下所示：

   - 首先，对覆盖进行评估。这意味着会始终为某些用户提供特定变体。已分配覆盖的用户的会话无益于启动或试验指标。
   - 然后，剩余流量可用于持续启动（如有）。如果启动正在进行，Launches（启动）部分中的表格会显示功能变体中的启动名称和启动流量拆分。在 Launches（启动）部分的右侧，Traffic（流量）指示器会显示分配给此启动的可用受众（覆盖后）的数量。未分配给此启动的其余流量会流向实验（如有），然后流向默认变体。
   - 然后，剩余流量可用于正在进行的实验（如有）。如果有实验正在进行，Experiments（实验）部分中的表格会显示实验名称和进度。在 Experiments（实验）部分的右侧，Traffic（流量）指示器会显示分配给此实验的可用受众（覆盖和启动后）的数量。将向未分配给启动或实验的其余流量提供功能的默认变体。

修改启动流量

您可以随时（包括启动期间）修改启动的流量分配。

如果您对同一功能同时进行启动和实验，则对功能流量的任何更改都会导致实验流量发生变化。这是因为可用于实验的受众是尚未分配给启动的总受众的一部分。增加启动流量会导致可用于实验的受众减少，减少启动流量或结束启动则会导致可用于实验的受众增加。

修改启动的流量分配

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含启动的项目名称。
4. 请选择 Launches（启动）选项卡。
5. 请选择启动的名称。

   请选择 Modify launch traffic（修改启动流量）。
6. 对于 Serve（服务），选择要分配给每个变体的新流量百分比。您也可以选择不向用户提供变体。当您更改这些值时，您可以在 Traffic summary（流量摘要）下查看对整体功能流量的更新影响。

   Traffic summary（流量摘要）显示了可用于此次启动的总流量数量，以及其中可分配给此次启动的数量。
7. 选择修改。

修改启动的未来步骤

您可以修改尚未进行的启动步骤配置，还可以为启动添加更多步骤。

修改启动步骤

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含启动的项目名称。
4. 请选择 Launches（启动）选项卡。
5. 请选择启动的名称。

   请选择 Modify launch traffic（修改启动流量）。

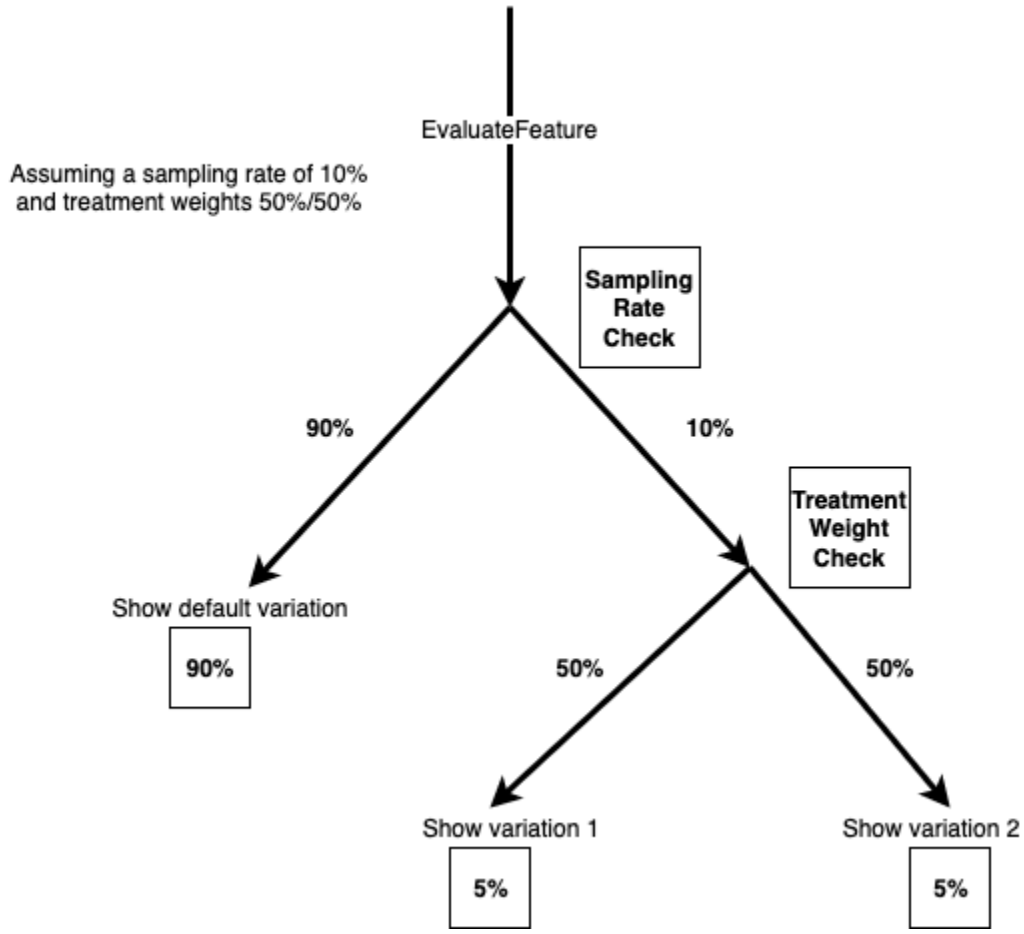
6. 选择 Schedule launch（计划启动）。
7. 对于尚未开始的任何步骤，您可以修改在实验中使用的可用受众百分比。您还可以修改其流量在变体中的分配方式。

   您可以通过选择 Add another step（添加其他步骤）为启动添加更多步骤。启动可具有最多五个步骤。
8. 选择修改。

修改实验流量

您可以随时（包括实验进行期间）修改实验的采样率。但是，在实验运行后，您无法更新处理权重。因此，您可以在实验运行后更改分配给实验的总流量，但不能更改每次处理的相对分配。如果您修改正在进行的实验的流量，我们建议您只增加流量分配，以免产生偏差。

下图显示在一项实验中如何将客户端流量分配给不同的变体。在此实验中，采样率为 10%，两种变体的处理权重各为 50%。



修改实验的流量分配

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Application monitoring（监控应用程序）、Evidently。
3. 请选择包含启动的项目名称。
4. 请选择 Experiments（实验）选项卡。
5. 请选择启动的名称。
6. 选择 Modify experiment traffic（修改实验流量）。
7. 输入百分比或使用滑块，指定向此实验分配的可用流量。可用流量为总受众减去分配给当前启动的流量（如有）。将向未分配给启动或实验的流量提供默认变体。
8. 选择修改。

停止启动

如果停止正在进行的启动，您将无法恢复或重启。此外，不会将其作为流量分配的规则进行评估，并且分配给此次启动的流量可用于功能的实验（如有）。否则，在启动停止后，将向所有流量提供默认变体。

永久停止启动

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含启动的项目名称。
4. 请选择 Launch（启动）选项卡。
5. 选择启动名称左侧的按钮。
6. 选择 Actions（操作）、Cancel launch（取消启动）或 Actions（操作）、Mark as complete（标记为完成）。

停止实验

如果您停止正在进行的实验，将无法恢复或重启。将向之前在实验中使用的流量部分提供默认变体。

如果试验未被手动停止并且超过结束日期，流量不会改变。分配给实验的流量部分仍然用于实验。要停止这种情况，并向实验流量提供默认变体，请将实验标记为完成。

如果您想停止实验，可以选择将其取消或将其标记为完成。如果将其取消，将在实验列表中显示 Cancelled（已取消）。如果选择将其标记为完成，将显示 Completed（已完成）。

永久停止实验

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含此实验的项目名称。
4. 请选择 Experiments（实验）选项卡。
5. 请选择实验名称左侧的按钮。
6. 请选择 Actions（操作）、Cancel experiment（取消实验）或 Actions（操作）、Mark as complete（标记为完成）。

将代码添加到应用程序

要使用 CloudWatch Evidently，您可以将代码添加到应用程序添，以向每个用户会话分配变体，并将指标发送到 Evidently。使用 CloudWatch Evidently `EvaluateFeature` 操作将变体分配给用户会话，然后使用 `PutProjectEvents` 操作将事件发送到 Evidently，用于计算启动或实验的指标。

当您创建变体或自定义指标时，CloudWatch Evidently 控制台会提供需要添加的代码示例。

有关端到端示例的信息，请参阅 教程：使用 Evidently 示例应用程序进行 A/B 测试。

使用 EvaluateFeature

在启动或实验中使用功能变体时，应用程序会使用 EvaluateFeature 操作为每个用户会话分配一个变体。将变体分配给用户即为评估事件。如果您调用此操作，会传递以下内容：

- Feature name（功能名称）– 必填。Evidently 会根据启动或实验的功能评估规则进行评估，并为实体选择变体。
- entityId – 必填。表示唯一的用户。
- evaluationContext – 可选。一个 JSON 对象，表示有关用户的其他信息。如果您已经创建了细分，Evidently 会使用此值在功能评估期间将用户与受众细分匹配。有关更多信息，请参阅 使用细分来聚焦受众。

  下面是您可以发送至 Evidently 的 `evaluationContext` 值的示例。

```
{
```

```
    "Browser": "Chrome",
    "Location": {
        "Country": "United States",
        "Zipcode": 98007
    }
}
```

粘性评估

CloudWatch Evidently 使用“粘性”评估。entityId 的单个配置、功能、功能配置 和 evaluationContext 始终会接收相同的变体分配。此变体分配仅在实体添加到覆盖或实验流量增加时更改。

功能配置包括以下内容：

- 功能变体
- 此功能当前运行的实验的变体配置（分配给每个变体的百分比）（如果有）。
- 此功能当前运行的启动的变体配置（如果有）。变体配置包括定义的区段覆盖（如果有）。

如果增加实验的流量分配，则之前分配给实验处理组的所有 entityId 都将继续接受相同的处理。根据为实验指定的变体配置，之前分配给对照组的所有 entityId 都可能被分配到实验处理组。

如果减少实验的流量分配，则 entityId 可能会从处理组转移到对照组，但不会进入不同的处理组。

使用 PutProjectEvents

要为 Evidently 编写自定义指标，请使用 PutProjectEvents 操作。以下是一个简单的有效负载示例。

```
{
    "events": [
        {
            "timestamp": {{$timestamp}},
            "type": "aws.evidently.custom",
            "data": "{\"details\": {\"pageLoadTime\": 800.0}, \"userDetails\":
 {\"userId\": \"test-user\"}}"
        }
    ]
}
```

entityIdKey 可以只是 entityId，您也可以将其重命名为其他任何名称，例如 userId。在实际事件中，entityId 可以是用户名、会话 ID 等等。

```
"metricDefinition":{
    "name": "noFilter",
    "entityIdKey": "userDetails.userId", //should be consistent with jsonValue in
 events "data" fields
    "valueKey": "details.pageLoadTime"
},
```

为了确保事件与正确的启动或实验相关联，您必须在调用 EvaluateFeature 和 PutProjectEvents 时传递相同的 entityId。请务必在调用 EvaluateFeature 之后调用 PutProjectEvents，否则数据会丢弃，并且 CloudWatch Evidently 不会使用。

PutProjectEvents 操作不要求将功能名称作为输入参数。这样一来，您可在多个实验中使用一个事件。例如，假设您调用 EvaluateFeature 并将 entityId 设置为 userDetails.userId。如果您有两个或更多运行中的实验，则可以从该用户的会话中获得单个事件，并为每个这些实验发出指标。为此，使用相同的 entityId 为每个实验调用一次 PutProjectEvents。

Timing

在应用程序调用 EvaluateFeature 之后，有一个小时的时间段，其中根据该评估限制来自 PutProjectEvents 的指标事件。如果在一小时之后发生了更多的事件，则不会限制这些事件。

但是，如果在初始调用的一小时时段内将相同的 entityId 用于新 EvaluateFeature 调用，则现在改为使用后一个 EvaluateFeature 的结果，并重新启动一个小时的计时器。仅在某些情况下才会发生此操作，例如在两此分配之间增加实验流量时，如前面的粘性评估部分所解释。

有关端到端示例的信息，请参阅 教程：使用 Evidently 示例应用程序进行 A/B 测试。

项目数据存储

Evidently 收集两种类型的事件：

- 评估事件与分配给用户会话的功能变体有关。Evidently 使用这些事件，生成可在 Evidently 控制台中查看的指标与其他实验和启动数据。

  您也可以选择将这些评估事件存储在 Amazon CloudWatch Logs 或 Amazon S3 中。
- 自定义事件用于通过单击和结算等用户操作生成指标。Evidently 不会提供存储自定义事件的方法。如果要将其保存，您必须修改应用程序代码以将其发送到 Evidently 之外的存储选项。

评估事件日志的格式

如果您选择将评估事件存储在 CloudWatch Logs 或 Amazon S3 中，则每个评估事件都将以下列格式存储为日志事件：

```
{
    "event_timestamp": 1642624900215,
    "event_type": "evaluation",
    "version": "1.0.0",
    "project_arn": "arn:aws:evidently:us-east-1:123456789012:project/petfood",
    "feature": "petfood-upsell-text",
    "variation": "Variation1",
    "entity_id": "7",
    "entity_attributes": {},
    "evaluation_type": "EXPERIMENT_RULE_MATCH",
    "treatment": "Variation1",
    "experiment": "petfood-experiment-2"
}
```

以下是关于上述评估事件格式的更多详细信息：

- 时间戳为以毫秒为单位的 UNIX 时间
- 此变体是分配给此用户会话的功能变体的名称。
- 实体 ID 是字符串。
- 实体属性是客户端发送的任意值的哈希值。例如，如果 `entityId` 映射为蓝色或绿色，然后您可以选择从关联和数据仓库的角度选择性地发送用户 ID、会话数据或任何您想要的其它内容。

用于在 Amazon S3 中存储评估事件的 IAM policy 和加密

如果您选择使用 Amazon S3 存储评估事件，则必须添加类似以下内容的 IAM policy，以允许 Evidently 将日志发布到 Amazon S3 存储桶。这是因为 Amazon S3 存储桶及其包含的对象是私有的，默认情况下它们不允许访问其他服务。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AWSLogDeliveryWrite",
            "Effect": "Allow",
```

```
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bucket_name/optional_folder/AWSLogs/account_id/
*",
            "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-
control"}}
        },
        {
            "Sid": "AWSLogDeliveryCheck",
            "Effect": "Allow",
            "Principal": {"Service": "delivery.logs.amazonaws.com"},
            "Action": ["s3:GetBucketAcl", "s3:ListBucket"],
            "Resource": "arn:aws:s3:::bucket_name"
        }
    ]
}
```

如果您将 Evidently 数据存储在 Amazon S3 中，您还可以选择使用带 AWS Key Management Service 密钥的服务器端加密 (SSE-KMS)，对其进行加密。有关更多信息，请参阅使用服务器端加密保护数据。

如果您使用 AWS KMS 的客户托管式密钥，您必须将以下内容添加到密钥的 IAM policy 中。这样一来，Evidently 便可以写入存储桶。

```
{
    "Sid": "AllowEvidentlyToUseCustomerManagedKey",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "delivery.logs.amazonaws.com"
        ]
    },
   "Action": [
       "kms:Encrypt",
       "kms:Decrypt",
       "kms:ReEncrypt*",
       "kms:GenerateDataKey*",
       "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Evidently 是如何计算结果的

您可以使用 Amazon CloudWatch Evidently A/B 测试作为数据驱动型决策的工具。在 A/B 测试中，用户被随机分配到对照组（也称为默认变体）或其中一个治疗组（也称为测试变体）。例如，对照组中的用户体验网站、服务或应用程序的方式可能与实验开始之前相同。同时，治疗组中的用户可能会体验到某种变化。

CloudWatch Evidently 在一项实验中最多支持五种不同的变体。Evidently 将流量随机分配给这些变体。这样，您可以跟踪每个组的业务指标（例如收入）和绩效指标（例如延迟）。Evidently 可执行以下操作：

- 将治疗组与对照组进行比较。（例如，比较在新的结账流程下收入是增加还是减少。）
- 表明治疗组和对照组之间观察到的差异是否为显著。为此，Evidently 提供了两种方法：频率论显著性水平和贝叶斯概率。

为什么要使用频率论方法和贝叶斯方法？

假设某个病例的治疗组与对照组相比没有效果，或者某个病例的治疗组与对照组效果相同（A/A 测试）。您仍然可以在数据中观察到治疗组和对照组之间存在细微差异。这是因为测试参加者由有限的用户样本组成，占网站、服务或应用程序所有用户的一小部分。通过频率论显著性水平和贝叶斯概率可以了解观察到的差异是显著的还是偶然的。

Evidently 考虑以下因素来确定观察到的差异是否显著：

- 差异有多大
- 测试中有多少样本
- 数据是如何分布的

Evidently 中的频率论分析

Evidently 使用顺序测试，这样可以避免常见的窥视问题，该问题是频率论统计方法中的一个常见陷阱。窥视是一种查看正在进行的 A/B 测试的结果的做法，其目的是停止测试并根据观察到的结果做出决定。有关顺序测试的更多信息，请参阅 Howard 等撰写的 时间均匀、非参数、非渐近置信序列 [《Ann. Statist.》（统计学年鉴）第 49 卷第 2 期，1055-1080 页，2021 年]。

因为 Evidently 的结果在任何时候都是有效的（随时有效结果），您可以在实验过程中窥视结果，且仍然可以得出合理的结论。这样可以降低一些实验成本，因为如果实验结果已具有显著性，则可以在计划时间之前提前停止实验。

Evidently 可生成随时有效的显著性水平，并为目标指标中测试变体与默认变体之间的差异生成随时有效的 95% 置信区间。实验结果中的 Result（结果）列表示测试变体表现，可以为以下各项之一：

- Inconclusive（不确定）– 显著性水平低于 95%
- Better（更好）– 显著性水平为 95% 或以上，并且满足以下条件之一：
  - 95% 置信区间的下限高于零，指标应增加
  - 95% 置信区间的上限低于零，指标应减小
- Worse（更差）– 显著性水平为 95% 或以上，出现以下情况之一：
  - 95% 置信区间的上限高于零，指标应增加
  - 95% 置信区间的下限低于零，指标应减小
- Best（最好）– 除默认变体外，该实验还有两个或多个测试变体，并且满足以下条件：
  - 该变体符合 Better（更好）名称条件
  - 满足以下条件：
    - 95% 置信区间的下限高于所有其他变体的 95% 置信区间的上限，指标应增加
    - 95% 置信区间的上限低于所有其他变体的 95% 置信区间的下限，指标应减小

Evidently 中的贝叶斯分析

使用贝叶斯分析，您可以计算出测试变体中的均值大于或小于默认变体中的均值的概率。Evidently 使用共轭先验对目标指标的均值进行贝叶斯推断。使用共轭先验，Evidently 可以更高效地推断出贝叶斯分析所需的后验分布。

Evidently 要等到实验结束日期才计算贝叶斯分析的结果。结果页显示以下内容：

- 增加的概率 – 测试变体中指标的均值比默认变体中的均值至少大 3% 的概率
- 减小的概率 – 测试变体中指标的均值比默认变体中的均值至少小 3% 的概率
- 不变概率 – 测试变体中指标的均值在默认变体中的均值的 ±3% 内的概率

Result（结果）列表示变体性能，可以是以下各项之一：

- Better（更好）– 增加概率至少为 90%，指标应增加，或者减小概率至少为 90%，指标应减小
- Worse（更差）– 减小概率至少为 90%，指标应增加，或者增加概率至少为 90%，指标应减小

在控制面板中查看启动结果

在实验进行期间和实验完成后，您可以查看实验的进度和指标结果。

查看启动的进度和结果

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含启动的项目名称。
4. 请选择 Launch（启动）选项卡。
5. 请选择启动的名称。
6. 要查看启动步骤和每个步骤的流量分配，请选择 Launch（启动）选项卡。
7. 要查看一段时间内分配给每个变体的用户会话数，并查看启动中每个变体的性能指标，请选择 Monitoring（监控）选项卡。

   此视图还会显示在启动期间是否有任何启动告警进入 ALARM 状态。
8. 要查看此次启动的变体、指标、告警和标签，请选择 Configuration（配置）选项卡。

在控制面板中查看实验结果

您可以在实验进行期间和实验完成后，查看实验的统计结果。实验结果在实验开始后 63 天内可用。由于 CloudWatch 数据保留策略，在此之后实验结果将不可用。

每个变体至少有 100 个事件，才会显示统计结果。

Evidently 在实验结束时会执行额外的离线 p 值分析。在实验期间使用的任何 p 值找不到统计显著性的某些情况下，离线 p 值分析可以检测统计显著性。

有关 CloudWatch Evidently 如何计算实验结果的更多信息，请参阅 Evidently 是如何计算结果的。

查看实验结果

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择包含此实验的项目名称。
4. 请选择 Experiments（实验）选项卡。

5. 请选择实验名称，然后选择 Results（结果）选项卡。
6. 通过 Variation performance（变体性能），您可以使用一个控件来选择要显示的实验统计数据。如果您选择多个统计数据，Evidently 会显示每个统计数据的图表和表格。

   每个图表和表格都会显示迄今为止的实验结果。

   每个图表都会显示以下结果。您可以使用图表右侧的控件来确定显示以下哪些项目：

   - 为每个变体记录的用户会话事件数。
   - 在图表顶部选择的每个变体的指标平均值。
   - 实验的统计意义。这会将图表顶部所选指标的差异与默认变体和其他每个变体进行比较。
   - 每个变体与默认变体之间所选指标的差异，存在 95% 的置信上限和下限。

   此表会显示每个变体的一行。对于每个非默认变体，Evidently 会显示其是否收到足够的数据以表明结果具有统计意义。它还会显示变体的统计值改进是否已达到 95% 的置信水平。

   最后，在 Result（结果）列中，Evidently 会根据此统计数据就哪个变体性能最佳或者结果是否不确定提出建议。

CloudWatch Evidently 如何收集和存储数据

Amazon CloudWatch Evidently 会收集和存储与项目配置相关的数据，以便客户运行实验和启动。该数据包括以下内容：

- 有关项目、功能、启动和实验的元数据
- 指标事件
- 评估数据

资源元数据存储在 Amazon DynamoDB 中。默认情况下，数据将使用 AWS 拥有的密钥 进行静态加密。这些密钥是 AWS 服务 拥有并管理以用于多个 AWS 账户 的 AWS KMS 密钥的集合。客户无法查看、管理或审计这些密钥的使用情况。客户也无需采取行动或更改计划以保护用于加密其数据的密钥。

有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS 拥有的密钥。

Evidently 指标事件和评估事件会直接发送到客户的地址。

传输中的数据将使用 HTTPS 自动加密。这些数据将传送到客户的地址。

您也可以选择将评估事件存储在 Amazon Simple Storage Service 或 Amazon CloudWatch Logs 中。有关如何保护这些服务中的数据的更多信息，请参阅启用 Amazon S3 默认存储桶加密和使用 AWS KMS 对 CloudWatch Logs 中的日志数据进行加密。

检索数据

您可以使用 CloudWatch Evidently API 检索数据。要检索项目数据，请使用 GetProject 或 ListProjects。

要检索功能数据，请使用 GetFeature 或 ListFeatures。

要检索启动数据，请使用 GetLaunch 或 ListLaunches。

要检索实验数据，请使用 GetExperiment、ListExperiments 或者 GetExperimentResults。

修改和删除数据

您可以使用 CloudWatch Evidently API 修改和删除数据。对于项目数据，请使用 UpdateProject 或 DeleteProject。

对于功能数据，请使用 UpdateFeature 或 DeleteFeature。

对于启动数据，请使用 UpdateLaunch 或 DeleteLaunch。

对于实验数据，请使用 UpdateExperiment 或 DeleteExperiment。

使用 Evidently 的服务相关角色

CloudWatch Evidently 使用 AWS Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Evidently 直接相关。服务相关角色由 Evidently 预定义，并包含服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 Evidently，因为您不必手动添加必要的权限。Evidently 定义其服务相关角色的权限，除非另外定义，否则只有 Evidently 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其它 IAM 实体的权限策略。

只有在首先删除服务相关角色的相关资源后，才能删除该角色。这将保护您的 Evidently 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找 Service-linked roles（服务相关角色）列中显示为 Yes（是）的服务。请选择 Yes 与查看该服务的服务相关角色文档的链接。

Evidently 的服务相关角色权限

Evidently 使用名为 AWSServiceRoleForCloudWatchEvidently 的服务相关角色 — 允许 CloudWatch Evidently 代表您的客户管理关联的 AWS 资源。

AWSServiceRoleForCloudWatchEvidently 服务相关角色信任以下服务代入角色：

- `CloudWatch Evidently`

名为 AmazonCloudWatchEvidentlyServiceRolePolicy 的角色权限策略允许 Evidently 对指定资源完成以下操作：

- 操作：对 Evidently 胖客户端执行 `appconfig:StartDeployment`、`appconfig:StopDeployment`、`appconfig:ListDeployments` 和 `appconfig:TagResource`。

必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅 IAM 用户指南中的服务相关角色权限。

创建 Evidently 的服务相关角色

无需手动创建服务相关角色。当您在 AWS Management Console、AWS CLI 或 AWS API 中开始使用 Evidently 胖客户端时，Evidently 会为您创建服务相关角色。

如果删除此服务相关角色，然后需要再次创建，可以使用相同流程在账户中重新创建此角色。当您开始使用 Evidently 胖客户端时，Evidently 会再次为您创建服务相关角色。

编辑 Evidently 的服务相关角色

Evidently 不允许您编辑 AWSServiceRoleForCloudWatchEvidently 服务相关角色。创建服务相关角色后，将无法更改角色名称，因为可能有多个实体引用该角色。但是可以使用 IAM 编辑角色说明。有关更多信息，请参阅 IAM 用户指南中的编辑服务相关角色。

删除 Evidently 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先清除服务相关角色的资源，然后才能手动删除它。您必须删除所有使用胖客户端的 Evidently 项目。

> **Note**
>
> 如果在您试图删除资源时 Evidently 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

要删除 AWSServiceRoleForCloudWatchEvidently 使用的 Evidently 资源

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，依次选择 Application monitoring（监控应用程序）、Evidently。
3. 在项目列表中，选中使用胖客户端的项目旁边的复选框。
4. 选择 Project actions（项目操作）、Delete project（删除项目）。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForCloudWatchEvidently 服务相关角色。有关更多信息，请参阅 IAM 用户指南中的删除服务相关角色。

Evidently 服务相关角色的受支持区域

Evidently 支持在服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 AWS 区域和终端节点。

CloudWatch Evidently 配额

CloudWatch Evidently 具有以下配额。

资源	默认限额
项目	每账户每区域 50 个 您可以请求提高限额。
分段	每账户每区域 500 个 您可以请求提高限额。
每个项目的配额	• 共 100 个功能

资源	默认限额
	• 共 500 个启动 • 50 个正在运行的启动 • 共 500 个实验 • 50 个正在运行的实验 对于所有这些配额，您可以请求增加配额。
API 配额（所有配额均针对每个区域）	• PutProjectEvents：在美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和欧洲（爱尔兰），每秒 1000 个事务（TPS）。所有其他区域为 200 TPS。 • EvaluateFeature：在美国东部（弗吉尼亚州北部）、美国西部（俄勒冈州）和欧洲（爱尔兰），1000 TPS。所有其他区域为 200 TPS。 • BatchEvaluateFeature：50 TPS • 创建、读取、更新、删除 (CRUD) API：在所有 CRUD API 中组合 10 TPS 对于所有这些配额，您可以请求增加配额。

教程：使用 Evidently 示例应用程序进行 A/B 测试

本节介绍如何使用 Amazon CloudWatch Evidently 进行 A/B 测试的教程。本教程涉及 Evidently 示例应用程序，其是一个简单的响应应用程序。示例应用程序将被配置为显示/不显示 `showDiscount` 功能。当向用户显示该功能时，购物网站上以 20% 的折扣显示价格。

除了向一些用户而非向其他用户显示折扣之外，在本教程中，您还设置 Evidently 以从两个变体收集页面加载时间指标。

> **Warning**
>
> 此场景需要 IAM 用户具有编程访问权限和长期凭证，这会带来安全风险。为帮助减轻这种风险，我们建议仅向这些用户提供执行任务所需的权限，并在不再需要这些用户时将其移除。必要时可以更新访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 更新访问密钥。

步骤 1：下载示例应用程序

首先下载 Evidently 示例应用程序。

下载示例应用程序

1. 从以下 Simple Storage Service (Amazon S3) 存储桶下载示例应用程序：

   ```
   https://evidently-sample-application.s3.us-west-2.amazonaws.com/evidently-sample-shopping-app.zip
   ```

2. 解压缩程序包。

步骤 2：添加 Evidently 端点并设置凭证

接下来，将 Evidently 的区域和端点添加到示例应用程序包中 `src` 目录下的 `config.js` 文件，如以下示例所示：

```
evidently: {
    REGION: "us-west-2",
    ENDPOINT: "https://evidently.us-west-2.amazonaws.com (https://evidently.us-west-2.amazonaws.com/)",
},
```

您还必须确保应用程序有调用 CloudWatch Evidently 的权限。

授予示例应用程序调用 Evidently 的权限

1. 联合到您的 AWS 账户。
2. 创建 IAM 用户并将 AmazonCloudWatchEvidentlyFullAccess 策略附加到该用户。
3. 记录 IAM 用户的访问密钥 ID 和秘密访问密钥，因为您在下一步中需要使用它们。
4. 在您本节早些时候修改的同一 `config.js` 文件中，输入访问密钥 ID 和秘密访问密钥的值，如以下示例所示：

   ```
   credential: {
      accessKeyId: "Access key ID",
      secretAccessKey: "Secret key"
   }
   ```

> **Important**
>
> 我们使用此步骤使示例应用程序尽可能简单，以便您尝试。我们不建议您将 IAM 用户凭证放到实际的生产应用程序中。我们建议您使用 Amazon Cognito 进行身份验证。有关更多信息，请参阅将 Amazon Cognito 与 Web 和移动应用程序集成。

步骤 3：为功能评估设置代码

当您使用 CloudWatch Evidently 评估功能时，必须使用 EvaluateFeature 操作为每个用户会话随机选择功能变体。此操作根据您在实验中指定的百分比，将用户会话分配给功能的每个变体。

为书店演示应用程序设置功能评估代码

1. 将客户端生成器添加到 `src/App.jsx` 文件，以便示例应用程序可以调用 Evidently。

```
import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
    endpoint,
    region,
) => {
    const credentials = {
        accessKeyId: config.credential.accessKeyId,
        secretAccessKey: config.credential.secretAccessKey
    }
    return new Evidently({
        endpoint,
        region,
        credentials,
    });
};
```

2. 将以下内容添加到 `const App` 代码部分以启动客户端。

```
  if (client == null) {
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
```

```
      );
```

3. 通过添加以下代码构建 evaluateFeatureRequest。此代码会预填充我们在本教程后面推荐的项目名称和功能名称。您可以替换自己的项目名称和功能名称，只要您也可以在 Evidently 控制台中指定这些项目名称和功能名称。

```
const evaluateFeatureRequest = {
              entityId: id,
              // Input Your feature name
              feature: 'showDiscount',
              // Input Your project name'
              project: 'EvidentlySampleApp',
            };
```

4. 添加代码以调用 Evidently 进行功能评估。发送请求时，Evidently 会随机分配查看或不查看 showDiscount 功能的用户会话。

```
client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
    }
    getPageLoadTime()
 })
```

步骤 4：为实验指标设置代码

对于自定义指标，请使用 Evidently 的 PutProjectEvents API 向 Evidently 发送指标结果。以下示例说明了如何设置自定义指标并向 Evidently 发送实验数据。

添加以下函数来计算页面加载时间，并使用 PutProjectEvents 向 Evidently 发送指标值。将以下函数添加到 Home.tsx 中并在 EvaluateFeature API 中调用此函数：

```
const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
    const pageLoadTimeData = `{
      "details": {
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
```

```
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
```

自您下载 App.js 文件并对其进行所有编辑之后，该文件应该如此处所示。

```
import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Switch } from "react-router-dom";
import AuthProvider from "contexts/auth";
import CommonProvider from "contexts/common";
import ProductsProvider from "contexts/products";
import CartProvider from "contexts/cart";
import CheckoutProvider from "contexts/checkout";
import RouteWrapper from "layouts/RouteWrapper";
import AuthLayout from "layouts/AuthLayout";
import CommonLayout from "layouts/CommonLayout";
import AuthPage from "pages/auth";
import HomePage from "pages/home";
import CheckoutPage from "pages/checkout";
import "assets/scss/style.scss";
import { Spinner } from 'react-bootstrap';

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
  region,
) => {
  const credentials = {
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
```

```
    region,
    credentials,
  });
};

const App = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [startTime, setStartTime] = useState(new Date());
  const [showDiscount, setShowDiscount] = useState(false);
  let client = null;
  let id = null;

  useEffect(() => {
    id = new Date().getTime().toString();
    setStartTime(new Date());
    if (client == null) {
      client = defaultClientBuilder(
        config.evidently.ENDPOINT,
        config.evidently.REGION,
      );
    }
    const evaluateFeatureRequest = {
      entityId: id,
      // Input Your feature name
      feature: 'showDiscount',
      // Input Your project name'
      project: 'EvidentlySampleApp',
    };

    // Launch
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
      }
    });

    // Experiment
    client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
      if(res.value?.boolValue !== undefined) {
        setShowDiscount(res.value.boolValue);
      }
      getPageLoadTime()
    })
```

```
    setIsLoading(false);
  },[]);

  const getPageLoadTime = () => {
    const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
    const pageLoadTimeData = `{
      "details": {
        "pageLoadTime": ${timeSpent}
      },
      "UserDetails": { "userId": "${id}", "sessionId": "${id}"}
    }`;
    const putProjectEventsRequest = {
      project: 'EvidentlySampleApp',
      events: [
        {
          timestamp: new Date(),
          type: 'aws.evidently.custom',
          data: JSON.parse(pageLoadTimeData)
        },
      ],
    };
    client.putProjectEvents(putProjectEventsRequest).promise();
  }
  return (
    !isLoading? (
    <AuthProvider>
      <CommonProvider>
        <ProductsProvider>
          <CartProvider>
            <CheckoutProvider>
              <Router>
                <Switch>
                  <RouteWrapper
                    path="/"
                    exact
                    component={() => <HomePage showDiscount={showDiscount}/>}
                    layout={CommonLayout}
                  />
                  <RouteWrapper
                    path="/checkout"
                    component={CheckoutPage}
                    layout={CommonLayout}
                  />
                  <RouteWrapper
```

```
                    path="/auth"
                    component={AuthPage}
                    layout={AuthLayout}
                  />
                </Switch>
              </Router>
            </CheckoutProvider>
          </CartProvider>
        </ProductsProvider>
      </CommonProvider>
    </AuthProvider> ) : (
      <Spinner animation="border" />
    )
  );
};

export default App;
```

每次用户访问示例应用程序时，都会向 Evidently 发送自定义指标进行分析。Evidently 会分析每个指标并在 Evidently 控制面板上实时显示结果。以下示例显示了指标有效负载：

```
[ {"timestamp": 1637368646.468, "type": "aws.evidently.custom", "data": "{\"details
\":{\"pageLoadTime\":2058.002058},\"userDetails\":{\"userId\":\"1637368644430\",
\"sessionId\":\"1637368644430\"}}" } ]
```

步骤 5：创建项目、功能和实验

然后，您可以在 CloudWatch Evidently 控制台中创建项目、功能和实验。

为本教程创建项目、功能和实验

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Application Signals、Evidently。
3. 请选择 Create project（创建项目）并填写字段。您必须使用 **`EvidentlySampleApp`** 作为项目名，以便示例正常工作。对于 Evaluation event storage（评估事件存储），选择 Don't store Evaluation events（不要存储评估事件）。

   填写字段后，选择 Create Project（创建项目）。

   有关更多详细信息，请参阅创建新 项目。

4. 创建项目后，在该项目中创建功能。将功能命名为 **showDiscount**。在此功能中，创建 **Boolean** 类型的两个变体。使用值 **disable** 将第一个变体命名为 **False**，然后使用值 **enable** 将第二个变体命名为 **True**。

   有关创建功能的更多信息，请参阅 向项目添加功能。

5. 完成功能创建后，请在项目中创建实验。将实验命名为 **pageLoadTime**。

   本实验将使用名为 pageLoadTime 的自定义指标，用以衡量被测试页面的页面加载时间。用于实验的自定义指标使用 Amazon EventBridge 创建而成。有关 EventBridge 的更多信息，请参阅什么是 Amazon EventBridge？。

   要创建该自定义指标，请在创建实验时执行以下操作：

   - 对于 Metric source（指标来源），请在 Metrics（指标）下选择 Custom metrics（自定义指标）。
   - 对于 Metric name（指标名称），请输入 **pageLoadTime**。
   - 对于 Goal（目标），请选择 Decrease（减少）。这表示我们希望该指标的值较低，以指明该功能变体最佳。
   - 对于 Metric rule（指标规则），请输入以下内容：
     - 对于实体 ID，请输入 **UserDetails.userId**。
     - 对于 Value key（值键），请输入 **details.pageLoadTime**。
     - 对于 Units（单位）中，请输入 **ms**。
   - 请选择 Add metric（添加指标）。

   对于 Audiences（受众），请选择 100% 以便所有用户都能进入实验。将变体之间的流量拆分设置为每个变体 50%。

   然后，请选择 Create experiment（创建实验）来创建实验。创建它后，只有您告知 Evidently 将其启动，它才会启动。

步骤 6：开始实验并测试 CloudWatch Evidently

最后的步骤是开始实验并开启示例应用程序。

开始教程实验

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Application Signals、Evidently。
3. 选择 EvidentlySampleApp 项目。
4. 请选择 Experiments（实验）选项卡。
5. 请选择 pageLoadTime 旁边的按钮，然后依次选择 Actions（操作）、Start experiment（开始实验）。
6. 请选择实验结束的时间。
7. 请选择 Start experiment（开始实验）。

   实验会立即开始。

然后，使用以下命令开启 Evidently 示例应用程序：

```
npm install -f && npm start
```

开启应用程序后，您将被分配到正在测试的两个功能变体之一。一个变体会显示“20% 折扣”，另一个变体不会显示。继续刷新页面以查看不同的变体。

> **Note**
>
> Evidently 包含粘性评估。功能评估是确定性的，这意味着对于相同的 `entityId` 和功能，用户将始终收到相同的变体分配。变体分配仅在实体被添加到覆盖或实验流量增加时更改。
> 但是，为了让您轻松使用示例应用程序教程，Evidently 会在您每次刷新页面时重新分配示例应用程序功能评估，以便您在无需添加覆盖的情况下即可体验两种变体。

故障排除

我们建议您使用 npm 版本 6.14.14。如果你看到有关构建或开启示例应用程序的任何错误，并且您正在使用不同版本的 npm，请执行以下操作。

安装 **npm** 版本 6.14.14

1. 使用浏览器连接到 https://nodejs.org/download/release/v14.17.5/。
2. 下载 node-v14.17.5.pkg 并运行此 pkg 安装 npm。

   如果您看到 `webpack not found` 错误，导航至 `evidently-sample-shopping-app` 文件夹并尝试以下操作：

a. 删除package-lock.json
b. 删除yarn-lock.json
c. 删除node_modules
d. 从 package.json 中删除 webpack 依赖项
e. 运行以下命令：

```
npm install -f && npm
```

网络监控

本部分中的主题描述了由 Amazon CloudWatch 互联网监测仪和 Amazon CloudWatch 网络监测仪提供的 CloudWatch 网络和互联网监控功能。这些服务可帮助您从运营角度了解托管在 AWS 上的应用程序的网络和互联网性能和可用性。

- 网络监测仪使用 AWS 从其全球网络足迹中捕获的连接数据来计算面向互联网流量的性能和可用性基准。您可以查看流量模式和运行状况事件的全局视图，轻松地深入了解事件信息。您还可以收到影响应用程序客户端的互联网运行状况事件的警报。此外，您可以使用互联网监测仪提供的见解，通过使用 Amazon CloudFront 或通过其他 AWS 区域 进行路由来探索客户体验的潜在改进。
- 网络监测仪使用完全托管的代理方法，使您能够跟踪和可视化混合网络连接的延迟和数据包丢失。要收集测量值并让网络监测仪为您的应用程序创建运行状况事件警报，您可以创建探测器，这些探测器将从托管在 AWS 上的资源发送到本地目标 IP 地址。您无需安装其他代理即可监控网络性能。与互联网监测仪一样，您可以设置警报和阈值，获取相应信息，从而帮助您快速排查问题，然后采取措施以改善最终用户体验。

主题

- 使用 Amazon CloudWatch 网络监测仪
- 使用 Amazon CloudWatch 网络监测仪

使用 Amazon CloudWatch 网络监测仪

Amazon CloudWatch 网络监测仪可让您了解互联网问题如何影响 AWS 上托管的应用程序与最终用户之间的互联网性能和可用性。其可以将您诊断互联网问题所需的时间从几天缩短到几分钟。Internet Monitor 使用 AWS 从其全球网络足迹中捕获的连接数据来计算面向互联网流量的性能和可用性基准。这与 AWS 用来监控互联网正常运行时间和可用性的数据相同。以这些测量数据为基准，当运行应用程序的不同地理位置的最终用户（客户端）遇到重大问题时，Internet Monitor 会引起您的重视。

在 Amazon CloudWatch 控制台中，您可以查看流量模式和运行状况事件的全局视图，并轻松地深入了解不同地理粒度（位置）的事件信息。您可以清晰地可视化显示影响，并查明受影响的位置和网络 [ASN，通常是互联网服务提供商（ISP）]。如果网络监测仪确定互联网可用性或性能问题是由特定 ASN 或 AWS 网络引起的，则会提供该信息。

网络监测仪的主要功能

- 网络监测仪提供了可帮助您改善最终用户体验的洞察和建议。您可以近乎实时地探索如何通过切换使用其他服务或通过不同的 AWS 区域 将流量重新路由到工作负载来改善应用程序的预计延迟影响。
- 网络监测仪会存储客户端位置和 ASN 对（即城市-网络）的互联网监测数据。网络监测仪还会为指向您的应用程序以及每个 AWS 区域和边缘站点的流量创建聚合 CloudWatch 指标。借助网络监测仪控制面板，您可以快速确定影响应用程序性能和可用性的因素，以便跟踪和解决问题。
- 网络监测仪还会向 CloudWatch Logs 和 CloudWatch Metrics 发布互联网监测数据，以支持使用 CloudWatch 工具来探索所监测应用程序流量的特定城市-网络数据。或者，您也可以将互联网测量数据发布到 Amazon S3。
- Internet Monitor 将运行状况事件发送到 Amazon EventBridge，以便您设置通知。如果问题是由 AWS 网络引起的，您还将自动收到 AWS Health Dashboard 通知，该通知将告知您 AWS 正在采取哪些步骤来缓解问题。

如何使用网络监测仪

要使用网络监测仪，您需要创建一个监测仪并将应用程序的资源（例如 VPC、网络负载均衡器、CloudFront 分配或 WorkSpaces 目录）与之关联，以便网络监测仪了解应用程序面向互联网的流量位置。然后，网络监测仪将从 AWS 发布特定于城市-网络的互联网测量数据，即客户端访问您的应用程序的位置和 ASN [通常是互联网服务提供商（ISP）]。有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的工作原理。要开始使用网络监测仪，请参阅 开始借助控制台使用 Amazon CloudWatch 网络监测仪。

内容

- 支持 Amazon CloudWatch 网络监测仪的 AWS 区域
- Amazon CloudWatch 网络监测仪的定价
- Amazon CloudWatch 网络监测仪的组件和术语
- Amazon CloudWatch 网络监测仪中的全球互联网天气地图
- Amazon CloudWatch 网络监测仪的工作原理
- Amazon CloudWatch 网络监测仪示例用例
- 网络监测仪跨账户可观测性
- 开始借助控制台使用 Amazon CloudWatch 网络监测仪
- 将 CLI 与 Amazon CloudWatch 网络监测仪结合使用的示例
- 使用 Internet Monitor 控制面板进行监控和优化
- 使用 CloudWatch 工具和网络监测仪查询接口探索您的数据
- 使用 Amazon CloudWatch 网络监测仪创建警报

- 将 Amazon CloudWatch 网络监测仪与 Amazon EventBridge 结合使用
- 排查 CloudWatch 日志和指标访问错误
- Amazon CloudWatch 网络监测仪的数据保护和数据隐私
- 适用于 Amazon CloudWatch 网络监测仪的 Identity and Access Management
- Amazon CloudWatch 网络监测仪的配额

支持 Amazon CloudWatch 网络监测仪的 AWS 区域

本节列出了支持 Amazon CloudWatch 网络监测仪的 AWS 区域。有关支持网络监测仪的区域（包括选择支持的区域）的最新列表，请参阅《Amazon Web Services 一般参考》中的 Amazon CloudWatch 网络监测仪端点和配额。

请注意，尽管监测仪可以包含多个区域中的资源，但是网络监测仪仅将监测仪的数据存储在您创建监测仪所在的 AWS 区域 中。

区域名称（选择支持）	区域
Africa (Cape Town)	af-south-1
亚太地区（香港）	ap-east-1
亚太地区（海得拉巴）	ap-south-2
亚太地区（雅加达）	ap-southeast-3
亚太地区（墨尔本）	ap-southeast-4
欧洲地区（米兰）	eu-south-1
欧洲（西班牙）	eu-south-2
欧洲（苏黎世）	eu-central-2
中东（巴林）	me-south-1
中东（阿联酋）	me-central-1

区域名称（默认支持）	区域
美国东部（俄亥俄）	us-east-2
美国东部（弗吉尼亚州北部）	us-east-1
美国西部（加利福尼亚北部）	us-west-1
美国西部（俄勒冈州）	us-west-2
亚太地区（孟买）	ap-south-1
亚太地区（大阪）	ap-northeast-3
亚太地区（首尔）	ap-northeast-2
亚太地区（新加坡）	ap-southeast-1
亚太地区（悉尼）	ap-southeast-2
亚太地区（东京）	ap-northeast-1
加拿大（中部）	ca-central-1
欧洲地区（法兰克福）	eu-central-1
欧洲地区（爱尔兰）	eu-west-1
欧洲地区（伦敦）	eu-west-2
欧洲地区（巴黎）	eu-west-3
欧洲（斯德哥尔摩）	eu-north-1
南美洲（圣保罗）	sa-east-1

Amazon CloudWatch 网络监测仪的定价

使用 Amazon CloudWatch 网络监测仪，无需预付费用或长期订阅。Internet Monitor 的定价包含两个部分：按监测的资源收费和按城市网络收费。城市网络是客户端借以访问应用程序资源的位置以

及客户端访问资源所使用的网络（ASN，例如互联网服务提供商或 ISP）。请注意，您还需要按标准 CloudWatch 价格为日志，以及您创建的任何其他指标、控制面板、警报或洞察付费。

创建监测仪时，您可以选择一定百分比的流量进行监测。为了帮助控制账单成本，您还可以设置要监测的城市网络最大数量的限制。您可以通过编辑监测仪随时更新要监测的流量百分比或城市网络的最大数量限制。包括前 100 个城市网络（每个账户的所有监测仪）。之后，您只需为监测的实际额外数量的城市网络付费，最多不超过最大数量。

您只需为自己监控的实际额外数量的城市网络付费，最高可达最大数量，前 100 个城市网络（每个账户的所有监控器）不收取任何费用。从您的每月账单中扣除相当于 100 个城市网络成本的固定金额。

例如，一家大型跨国公司可以选择监测其 100% 面向互联网的流量，并为一台监测仪和一种资源设置最多 5 万个城市网络。假设流量达到 5 万个城市网络，则其账单的这一部分费用约为 2700 美元/月。对于另一家位于较少地理区域，且一台显示器具有一种资源和 200 个城市网络的公司来说，这部分账单约为每月 13 美元。有关更多信息，请参阅 选择一个城市-网络最大限制。

您可以使用定价计算器尝试不同的选项。要了解定价选项，请在 CloudWatch 定价计算器页面上向下滚动到 Internet Monitor。

有关网络监测仪和 CloudWatch 定价的更多信息，请参阅 Amazon CloudWatch 定价页面。

Amazon CloudWatch 网络监测仪的组件和术语

Amazon CloudWatch 网络监测仪使用或引用了以下内容。

监控

监测仪包括单个应用程序的资源，您可以查看这些应用程序的互联网性能和可用性测量数据，以及获取有关此应用程序的运行状况事件警报。为应用程序创建监测仪时，您可以为此应用程序添加资源以定义 Internet Monitor 要监测的城市（位置）。Internet Monitor 使用您添加的应用程序资源的流量模式，以便发布特定于与应用程序通信的位置和 ASN（通常是互联网服务提供商或 ISP）的互联网性能和可用性测量数据。换句话说，您添加的资源创建了您想要 Internet Monitor 监测并为之发布测量数据的城市网络范围。

已添加到监测仪的资源（“受监控的资源”）

添加到监测仪的资源是网络监测仪中“受监控的资源”。即：

- 您在区域中添加的每个 VPC 都是受监测的资源。在您添加 VPC 时，网络监测仪会监控 VPC 中任何面向互联网的应用程序的流量，例如，托管在 Amazon EC2 实例上、网络负载均衡器背后或 AWS Fargate 容器中的应用程序。
- 您在区域中添加的每个网络负载均衡器都是受监测的资源。

- 您在区域中添加的每个 WorkSpaces 目录都是受监测的资源。
- 您添加的每个 CloudFront 分配都是受监测的资源。

自治系统号（ASN）

在 Internet Monitor 中，ASN 通常指互联网服务提供商（ISP），例如 Verizon 或 Comcast。ASN 是客户端用于访问互联网应用程序的网络提供商。自治系统（AS）是一组互联网可路由的互联网协议（IP）前缀，属于一个网络或一组网络，这些网络都由一个组织管理、控制和监督。

城市网络（位置和 ASN）

城市网络是客户端访问应用程序资源的位置（例如城市）以及客户端访问资源所使用的 ASN [通常是互联网服务提供商（ISP）]。为了帮助控制账单成本，您可以为网络监测仪的每个监测仪设置要监控的城市网络最大数量的限制。您只需为监测的实际数量的城市网络付费，最多不超过最大数量。有关更多信息，请参阅选择城市网络最大限制。

互联网测量数据

网络监测仪还会每五分钟将所监测应用程序流量的前 500 个城市-网络的互联网监测数据发布到 CloudWatch Logs 中的日志文件。

这些测量数据量化了应用程序城市网络的性能分数、可用性分数、传输的字节数（输入和输出字节数）和往返时间。这些是针对特定于 VPC、网络负载均衡器、CloudFront 分配或 WorkSpaces 目录的城市网络测量数据。或者，您可以选择将所有受监测城市网络（最多 50 万个城市网络的服务限制）的互联网测量数据和事件发布到 Amazon S3 存储桶。

指标

Internet Monitor 为传输至应用程序的全球流量和传输至每个 AWS 区域 的全球流量生成 CloudWatch Metrics 的聚合指标。有关更多信息，请参阅 将 CloudWatch Metrics 与 Amazon CloudWatch 网络监测仪结合使用。

运行状况事件

Internet Monitor 会创建运行状况事件，提醒您注意影响应用程序的特定问题。Internet Monitor 可在全球范围内检测互联网问题，例如网络延迟增加。然后，它会使用 AWS 全球基础设施足迹中的互联网历史测量数据来计算当前问题对您的应用程序的影响，并创建运行状况事件。默认情况下，网络监测仪会根据总体影响和本地影响阈值创建运行状况事件。有关配置阈值的更多信息，请参阅更改运行状况事件阈值。

每个运行状况事件都包含有关受影响的城市网络的信息。您可以在 CloudWatch 控制台中查看运行状况事件，也可以将 AWS 开发工具包或 AWS CLI 与 Internet Monitor API 操作结合使用来查看运行状况事件。Internet Monitor 还会发送 Amazon EventBridge 运行状况事件通知。有关更多信息，请参阅网络监测仪何时会创建和解决运行状况事件。

网络事件

网络监测仪在可供所有 AWS 客户使用的互联网天气地图上，显示有关最近的全球运行状况事件（称为互联网事件）的信息。您无需在网络监测仪中创建监控即可查看互联网天气地图。与运行状况事件不同的是，互联网事件并非针对个人客户或其应用程序流量。有关更多信息，请参阅 Amazon CloudWatch 网络监测仪中的全球互联网天气地图。

阈值

网络监测仪会根据总体影响阈值和本地阈值创建运行状况事件。您可以更改默认阈值并配置其他选项，例如关闭本地阈值。有关配置阈值的更多信息，请参阅更改运行状况事件阈值。

性能和可用性分数

通过分析 AWS 收集的数据，Internet Monitor 可以检测到应用程序的互联网性能和可用性与 Internet Monitor 计算的估算基准相比何时有所下降。为了更清晰地看到这些下降值，Internet Monitor 以分数的形式向您报告信息。性能分数表示未出现性能下降的流量的估计百分比。同样，可用性分数表示未出现可用性下降的流量的估计百分比。有关更多信息，请参阅 AWS 如何计算性能和可用性分数。

传输的字节和传输的受监测字节

传输的字节是 AWS 中应用程序与客户端访问应用程序所在的城市网络（即位置和 ASN，通常是互联网服务提供商）之间的传入和传出流量的总字节数。传输的受监测字节是类似的指标，但仅包括受监测流量的字节。

往返时间

往返时间（RTT）是指从客户端用户发出请求到向用户返回响应所需的时间。当跨客户端位置（城市或其他地理区域）汇总 RTT 时，将根据每个客户端位置产生的应用程序流量大小对该值进行加权。

Amazon CloudWatch 网络监测仪中的全球互联网天气地图

Amazon CloudWatch 网络监测仪显示全球互联网天气地图，供所有 AWS 客户使用。要查看地图，请在 Amazon CloudWatch 控制台中导航到网络监测仪。

该地图突出显示了世界各地影响 AWS 客户的互联网事件（“中断”），以及存在性能或可用性问题的特定城市和网络（ASN，通常是互联网服务提供商）。互联网天气地图包括过去 24 小时的互联网事件。

您无需在网络监测仪中创建监控即可查看互联网天气地图。与网络监测仪中的运行状况事件不同的是，互联网事件并非针对个人客户或其应用程序流量。

在互联网天气地图上，您可以选择一个互联网事件来了解该事件的详细信息。对于互联网事件，您可以查看开始时间、结束时间（如果事件已结束）、当前状态（处于活动或已解决状态）和中断问题类型（可用性或性能）。要详细了解如何创建互联网天气地图及其包含的内容，请参阅全球互联网天气地图常见问题解答。

要查看和使用特定于应用程序流量和客户端位置的详细信息，您可以在网络监测仪中为应用程序轻松设置监控。然后，您将看到（当前和历史）性能和可用性模式和事件，并获得仅针对您的应用程序占用空间和客户而量身定制的运行状况事件警报。互联网天气地图为您提供总体视图，而特定的监控则将信息筛选为仅与您的应用程序相关的测量值和详细信息。借助监控，您还可以浏览历史指标并获取建议，以改善应用程序的客户体验。要了解更多信息，请参阅 开始借助控制台使用 Amazon CloudWatch 网络监测仪。

Amazon CloudWatch 网络监测仪的工作原理

本部分提供有关 Amazon CloudWatch 网络监测仪工作原理的信息。其中包括说明 AWS 如何收集用于检测互联网连接问题的数据，以及如何计算性能和可用性分数。

内容

- 网络监测仪如何只关注您的应用程序流量占用量
- AWS 如何衡量连接问题和计算测量值
- 网络监测仪中地理位置的准确性
- 网络监测仪何时会创建和解决运行状况事件
- 运行状况事件的报告时间
- 网络监测仪如何处理 IPv4 和 IPv6 流量
- 网络监测仪如何选择要包含的城市网络子集
- 如何创建全球互联网天气地图（常见问题解答）

网络监测仪如何只关注您的应用程序流量占用量

Internet Monitor 专注于监测您的 AWS 资源用户访问的互联网子集，而不是像其他工具那样从世界的各区域广泛监测您的网站。这也是一种经济高效的解决方案，大小公司都能负担得起。

Internet Monitor 使用了同样强大的探测器和问题检测算法，AWS 通过在 Internet Monitor 中创建运行状况事件，在内部利用这些探测器和算法来提醒您影响应用程序的连接问题。然后，根据您的应用程序资源，Internet Monitor 会覆盖从您的活跃观众那里创建的流量配置文件，从而使您能够访问生成的性能和可用性地图。

利用这些信息，Internet Monitor 可以只向您显示相关事件（即来自您有活跃观众的地方的事件），以及这些事件对您的整体观众量的影响。因此，一个事件有多大影响（按百分比计算）取决于您在全球范围内的总流量。

网络监测仪会存储客户端位置和 ASN 对（即城市-网络）的互联网监测数据。网络监测仪还会为指向您的应用程序以及每个 AWS 区域和边缘站点的流量创建聚合 CloudWatch 指标。

此外，网络监测仪还会每五分钟将向每个监测仪发送流量的前 500 个城市-网络的互联网监测数据发布到 CloudWatch Logs，以支持使用 CloudWatch 工具和其他方法来处理您的数据。或者，您可以选择将所有受监测城市网络（最多 50 万个城市网络的服务限制）的互联网测量数据发布到 Amazon S3 存储桶。有关更多信息，请参阅 在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3。

Internet Monitor 的优点包括：

- 使用 Internet Monitor 不会给 AWS 上托管的应用程序带来额外的负载或成本。
- 您无需在客户端资源或应用程序中包含性能测量代码。
- 您可以了解您的应用程序所连接的互联网的性能和可用性，包括“最后一英里”信息。

请注意，因为 Internet Monitor 根据您的 AWS 资源创建测量数据，所以 Internet Monitor 仅创建特定于您的应用程序流量的事件。全球互联网问题一般不会报告。此外，当服务位置为 AWS 区域时，发出的测量数据和事件旨在表示区域级别的连接，并不能准确表示最终用户位置与可用区之间的连接。

AWS 如何衡量连接问题和计算测量值

Amazon CloudWatch 网络监测仪通过自治系统号（ASN）[通常是互联网服务提供商（ISP）] 来监测不同 AWS 区域 之间以及 Amazon CloudFront 入网点（PoP）到不同客户端位置的互联网连接数据。AWS 操作人员每天在内部使用这些连接数据来主动检测全球互联网的连接问题。

对于每个 AWS 区域，我们知道互联网的哪些部分与该区域通信，并执行以下操作：

- 我们通过 30 天的滚动窗口积极监控互联网的这些部分。
- 我们使用网络和更高级别的协议探测器，包括入站和出站探测。

AWS 具有主动和被动探测器，可测量从每个 AWS 区域 以及 CloudFront 服务到整个互联网的第 90 个百分位数的延迟（性能）和可到达性（可用性）。它会监控服务与客户位置之间的异常连接模式，然后以警报形式报告给客户。

有关详细信息，请参阅下面几节：

- 计算可用性和 RTT
- 计算性能和可用性分数

- 计算 TTFB 和 RTT（延迟）
- 区域和可用区监测数据和聚合

计算可用性和 RTT

往返时间（RTT）是指从用户发出请求到向用户返回响应所需的时间。当跨最终用户位置汇总往返时间数据时，将根据每个最终用户位置驱动的流量对该值进行加权。

例如，如果有两个最终用户位置，一个使用 5 毫秒 RTT 提供 90% 的流量，另一个使用 10 毫秒 RTT 提供 10% 的流量，则 RTT 汇总结果是 5.5 毫秒（计算过程：5 毫秒 * 0.9 + 10 毫秒 * 0.1）。

请注意，测量最终用户端延迟的资源存在差异。对于网络监测仪的延迟测量数据，VPC、网络负载均衡器和 WorkSpaces 目录不包含最终用户端的延迟。

计算性能和可用性分数

AWS 拥有大量关于 AWS 服务与不同城市网络（位置和 ASN）之间的互联网性能和可用性的历史数据。通过对数据进行统计分析，网络监测仪可以检测到您的应用程序的互联网性能和可用性与其估算的基准相比何时有所下降。为了更容易看到这些下降，它会以运行状况分数（性能分数和可用性分数）的形式向您报告该信息。

运行状况分数是按不同的粒度计算的。我们以最精细的粒度计算某个地理区域（例如某个城市或城域）和 ASN（城市网络）的运行状况分数。我们还将单独的运行状况分数汇总为监测仪中应用程序的总体运行状况分数。如果您在查看性能或可用性分数时未以任何特定的地理位置或服务提供商进行筛选，则网络监测仪会提供总体运行状况分数。

在指定的时间段内，总体运行状况分数涵盖整个应用程序。当应用程序的城市-网络对性能或可用性分数在应用程序内达到或低于相应的性能或可用性运行状况事件阈值时，网络监测仪会触发运行状况事件。默认情况下，总体性能和可用性阈值均为 95%。网络监测仪还会根据您配置的值，根据本地阈值（如果该选项处于默认启用状态）创建运行状况事件。要了解有关配置运行状况事件阈值的更多信息，请参阅更改运行状况事件阈值。

但是，当您浏览监控和日志文件中的信息以调查问题和了解更多信息时，可以按特定的城市（位置）、网络（ASN 或互联网服务提供商）或同时按城市和网络进行筛选。这样就可以使用筛选条件来查看不同城市、ASN 或城市-网络对的运行状况分数，具体取决于您选择的筛选条件。

- 可用性分数表示未出现可用性下降的流量的估计百分比。Internet Monitor 可估算与观察到的总流量和可用性指标测量值相比下降的流量百分比。例如，最终用户/服务位置对的可用性分数为 99%，相当于其流量的 1% 出现了可用性下降。
- 性能分数表示未出现性能下降的流量百分比。例如，最终用户/服务位置对的性能分数为 99%，相当于其流量的 1% 出现了性能下降。

计算 TTFB 和 RTT（延迟）

首字节时间（TTFB）是指从客户端发出请求到其从服务器接收到首字节信息的时间。AWS 在计算 TTFB 时会测量从 Amazon EC2 或 Amazon CloudFront 到网络监测仪测量节点（包括节点的最终用户端）的时间。也就是说，对于 EC2 的 TTFB，网络监测仪会测量从用户到 Amazon EC2 区域的时间，对于 CloudFront 的 TTFB，网络监测仪会测量从用户到 CloudFront 的时间。

对于往返时间（RTT），网络监测仪会测量从公有 IP 地址所反映的城市-网络（即客户端位置和 ASN，后者通常是互联网服务提供商）到 AWS 区域 的时间。这意味着，对于从网关或 VPN 之后访问互联网的用户，网络监测仪不会测量最终用户端的延迟。

请注意，测量最终用户端延迟的资源存在差异。对于网络监测仪的延迟测量数据，VPC、网络负载均衡器和 WorkSpaces 目录不包含最终用户端的延迟。

在 CloudWatch 控制面板上，在流量洞察选项卡中的流量优化建议部分中，网络监测仪会包含平均 TTFB 信息，旨在帮助您评估应用程序不同设置的选项，从而提高性能。

区域和可用区的测量和聚合

尽管网络监测仪聚合了区域级别的测量值并分享了影响，但它计算的是可用区（AZ）级别的影响。这意味着，如果在一个事件中，只有一个可用区受到影响，并且您的大部分流量都流经该可用区，则您的流量确实会受到影响。但是，对于同一事件，如果您的应用程序流量没有流经受影响的可用区，则您不会看到影响。

请注意，这只适用于不是 WorkSpaces 目录的资源。WorkSpaces 目录仅在区域级别进行衡量。

网络监测仪中地理位置的准确性

对于位置信息，网络监测仪使用 MaxMind 提供的 IP-地理位置数据。网络监测仪测量数据中位置信息的准确性，取决于 MaxMind 数据的准确性。

请注意，对于美国以外的地点，`Metro` 级别测量可能不准确。

网络监测仪何时会创建和解决运行状况事件

网络监测仪根据当前设置的阈值为您监测的应用程序流量创建和关闭运行状况事件。网络监测仪具有默认阈值配置，您也可以设置自己的阈值配置。网络监测仪确定连接问题对应用程序的总体影响，以及对应用程序具有客户端的本地区域的影响，并在超过阈值时创建运行状况事件。

对于通过 AWS 提供给服务的网络流量，Internet Monitor 根据有关互联网性能和可用性的历史数据计算连接问题对客户端位置的影响。它根据客户使用您的应用程序的 ASN 和服务的地理位置，应

用与您的应用程序相关的信息：受影响的城市网络对。位置由您添加到监测仪的资源确定。然后，网络监测仪使用统计分析来检测何时出现性能和可用性下降，从而影响应用程序的客户端体验。

网络监测仪计算的性能和可用性分数以未出现下降的流量百分比表示。影响与此相反：它表示问题对客户的最终用户造成了多大的影响。例如，如果表示全球可用性下降的值为 93%，则相应的影响将为 7%。

当应用程序的城市-网络对性能或可用性分数在全球范围内达到或低于相应的性能或可用性运行状况事件阈值时，这会导致网络监测仪生成运行状况事件。默认情况下，性能和可用性阈值均为 95%。达到或低于阈值的值是累积性的，因此这可能意味着几个较小的事件合计达到阈值百分比，也可能意味着单个事件达到或低于阈值水平。

只要触发事件的性能或可用性分数等于或低于总体影响的相应运行状况事件阈值百分比，运行状况事件就会保持活动状态。当触发事件的分数或合计分数超过阈值时，网络监测仪会解决运行状况事件。

网络监测仪还根据本地阈值和问题所影响的总流量的百分比创建运行状况事件。您可以配置本地阈值的选项，也可以完全关闭本地阈值。

要了解有关配置运行状况事件阈值的更多信息，请参阅更改运行状况事件阈值。

运行状况事件的报告时间

Internet Monitor 使用聚合器收集有关互联网问题的所有信号，以便在几分钟之内在监视器中创建运行状况事件。

Internet Monitor 会尽量分析运行状况事件的起源，以确定该事件是由 AWS 还是 ASN 引起。事件解决后，会继续进行运行状况事件分析。Internet Monitor 可以在长达一个小时的时间内使用新信息更新事件。

网络监测仪如何处理 IPv4 和 IPv6 流量

如果您通过任何 IP 系列（IPv4 或 IPv6）向网络提供流量，网络监测仪将仅通过 IPv4 测量网络的运行状况，并向您显示运行状况事件以及可用性和性能指标。如果您提供来自双堆栈资源（如双堆栈 CloudFront 分发）的流量，则只有在 IPv4 流量与 IPv6 流量存在相同的资源问题时，网络监测仪才会引发运行状况事件，并显示性能分数或可用性分数下降。

请注意，网络监测仪传入总字节数和传出总字节数指标可以准确反映所有互联网流量（IPv4 和 IPv6）。

网络监测仪如何选择要包含的城市网络子集

当您为监测仪监控的城市网络数量设置最大限制或选择要监控的流量百分比时，网络监测仪会根据最近的最高流量选择要包含（监控）的城市网络。

例如，如果您将城市网络的最大限制设置为 100，网络监测仪会根据您的应用程序在最近一小时内的流量（最多）监控 100 个城市网络。具体而言，网络监测仪会监控在最近一小时窗口之前的最近一小时窗口内流量最多的前 100 个城市网络。

为了说明这一点，假设当前时间为下午 2:30。在这种情况下，您在监测仪中看到的流量在下午 1:00 至下午 2:00 之间捕获，网络监测仪用来确定前 100 个城市网络的流量测量值在下午 12:00 至下午 1:00 之间捕获。

如何创建全球互联网天气地图（常见问题解答）

Amazon CloudWatch 网络监测仪的互联网天气地图可在网络监测仪控制台上提供给所有经身份验证的 AWS 客户。本节包含有关如何创建，以及如何使用互联网天气地图的详细信息。

什么是网络监测仪的互联网天气地图？

互联网天气地图直观显示了世界各地的互联网问题。它突出显示了受影响的客户位置，即城市加 ASN（通常是互联网服务提供商）。该地图显示了可用性和性能问题的组合，这些问题最近影响了全球热门客户位置和 AWS 服务的客户互联网体验。

地图数据来自何处？

这些数据基于对互联网的主动和被动探测的组合。要了解有关网络监测仪如何测量数据的更多信息，可以阅读 AWS 如何测量连接问题章节。

地图多久更新一次？

互联网天气地图每 15 分钟更新一次。

为防止中断，会跟踪哪些网络？

AWS 跟踪世界各地的网络，这些网络代表客户用于与 AWS 建立互联网连接的重要 IP 前缀。我们将客户位置中断的范围限定在 AWS 网络发送和接收流量的用量最高者。

哪些因素决定了互联网事件是否包含在地图上？

以下是我们用来确定是否将互联网事件包含在互联网天气地图上的一些高级标准：

- AWS 检测到存在可用性或性能事件。
- 如果该事件很短（例如，其持续时间不到 5 分钟），我们会忽略该事件。
- 然后，如果事件发生在被归类为用量最高者的客户位置，则被视为中断。

互联网天气地图使用哪些阈值？

对于互联网天气地图来说，确定中断的阈值不是静态的。网络监测仪根据检测到与预期值的偏差来确定事件的构成。通过查看网络监测仪如何确定何时为使用该服务创建的监控创建运行状况事

件，您可以了解有关其工作原理的更多信息。创建监控时，网络监测仪会生成特定于您自己的应用程序流量的互联网流量运行状况测量值。网络监测仪还针对影响应用程序互联网流量的问题，向您发出运行状况事件提醒。

我能用这些数据做什么？

互联网天气图提供过去 24 小时内世界各地发生的主要互联网事件的快速摘要。它可以帮助您体验互联网监控体验，而无需将自己的互联网流量加载到网络监测仪。要充分利用 AWS 互联网监控功能的潜力，并针对 AWS 上托管的应用程序和服务进行个性化设置，您可以在网络监测仪中创建监控。

创建监控后，可以让网络监测仪识别影响应用程序客户端的特定互联网路径，这样您就可以访问能帮助您改善客户端体验的特性和功能。您还将主动获知专门影响您的应用程序流量和客户端的新互联网问题。

我如何才能获得有关事件的更多详细信息？

单击地图上的中断可查看详细信息，包括事件开始和结束时间、受影响的城市和 ASN 以及问题类型（即性能问题还是可用性问题）。

要获取有关事件的更多详细信息，以及获取应用程序流量的自定义测量值，请在网络监测仪中创建监控。

Amazon CloudWatch 网络监测仪示例用例

在这一部分中，我们介绍了几个具体的示例，并提供了包含更多详细信息的博客文章链接。这些示例说明了如何使用 Amazon CloudWatch 网络监测仪的功能，来帮助监测应用程序并改善用户体验。

设置提醒并决定要执行的操作

您可以使用网络监测仪来深入了解一段时间内的平均互联网性能指标，以及城市-网络（客户端位置和 ASN，后者通常是互联网服务提供商）的运行状况问题。使用网络监测仪，您可以确定影响 Amazon 虚拟私有云（VPC）、网络负载均衡器、Amazon WorkSpaces 或 Amazon CloudFront 上托管应用程序的最终用户体验的事件。

创建监测仪后，您可以通过多种选项来确定如何接收有关网络监测仪运行状况事件的提醒。其中包括使用事件指标或 Amazon EventBridge 规则来筛选运行状况事件的 CloudWatch 警报通知。您可以根据警报为通知或操作选择不同选项，例如 AWS SMS 通知或对 CloudWatch 日志组的更新。

要查看带详细指南的示例，请参阅下列博客文章：Amazon CloudWatch 网络监测仪简介。

识别延迟问题并改进 TTFB，以提升多人游戏体验

借助网络监测仪，您可以快速查明在全球云游戏应用程序中，全球玩家遇到延迟问题的所在，并深入分析如何提高性能。通过确定当前大多数玩家遇到最慢首字节时间（TTFB）的位置，您就可以知道如何减少延迟，让您最大的玩家群更加满意。

现在，当您准备为游戏部署下一台 EC2 服务器时，请选择网络监测仪建议的 AWS 区域，这会降低高延迟、玩家群体庞大的区域的 TTFB。

有关在此应用场景中设置和使用网络监测仪的详细信息，请参阅下列博客文章：使用 Amazon CloudWatch 网络监测仪获得更好的游戏体验。

网络监测仪跨账户可观测性

借助网络监测仪跨账户可观测性，您可以监控单个 AWS 区域 中跨多个 AWS 账户的应用程序。

您可以使用 Amazon CloudWatch Observability Access Manager 将一个或多个 AWS 账户设置为监控账户。通过在监控账户中创建接收器，监控账户将能够查看源账户中的数据。接收器是表示监控账户中连接点的资源。对于网络监测仪，资源连接点就是一个监测仪。您可以使用接收器来创建从源账户指向监控账户的链接。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

所需的 资源

为保证 CloudWatch Application Insights 跨账户可观测性正常运行，请务必通过 CloudWatch Observability Access Manager 共享以下遥测类型。

- 网络监测仪中的监测仪
- Amazon CloudWatch 中的指标
- 在 Amazon CloudWatch Logs 中记录组

开始借助控制台使用 Amazon CloudWatch 网络监测仪

要开始使用 Amazon CloudWatch 网络监测仪，您必须在网络监测仪中为您的应用程序创建监测仪，为此您需要添加应用程序要使用的 AWS 资源并设置一些配置选项。本章说明了在控制台中添加监测仪的过程。另外还有一个章节包含了有关网络监测仪中资源的更多详细信息，其他的章节则介绍了您可以或必须为监测仪配置的不同选项以及相关限制。

内容

- 使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪

- 向监测仪添加资源
- 选择要监测的应用程序流量百分比
- 选择一个城市-网络最大限制
- 在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3
- 使用 Internet Monitor 监测仪
- 编辑或删除网络监测仪
- 更改网络监测仪的运行状况事件阈值
- 使用 Amazon VPC 添加或创建 Amazon CloudWatch 网络监测仪
- 通过 CloudFront 添加或创建 Amazon CloudWatch 网络监测仪

使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪

您需要在 Amazon CloudWatch 网络监测仪中为您的应用程序创建监测仪，为此您需要添加应用程序要使用的 AWS 资源，然后设置一些配置选项。您添加的资源 [Amazon 虚拟私有云（VPC）、网络负载均衡器（NLB）、CloudFront 分配或 WorkSpaces 目录] 会为网络监测仪提供信息，以映射应用程序的互联网流量信息。创建监测仪后，请等待 15-30 分钟以生成特定于应用程序使用位置的流量配置文件。然后，您可以使用网络监测仪或其他工具，来可视化显示和探索有关客户端使用情况的性能和可用性。这些工具使用监测仪采集并发布到（例如）的 CloudWatch Logs应用程序流量测量值，来为您提供相关的见解。

通常，最简单的方法是在 Internet Monitor 中为一个应用程序创建一个监测仪。在同一个监测仪中，您可以按不同的位置和 ASN（通常为互联网服务提供商）或其他信息，在网络监测仪日志文件中搜索测量数据和指标并进行排序。例如，没有必要为不同区域的应用程序创建单独的监测仪。

此处的步骤将引导您使用控制台设置监测仪。要查看将 AWS Command Line Interface 与 Internet Monitor API 操作结合使用的示例以创建监测仪、查看事件等，请参阅 将 CLI 与 Amazon CloudWatch 网络监测仪结合使用的示例。

使用控制台创建监测仪

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 选择 Create monitor（创建监控）。
4. 对于 Monitor name（监视器名称），请在 Internet Monitor 中输入要用于此监视器的名称。
5. 选择 Add resources（添加资源），然后选择资源以设置 Internet Monitor 用于此监视器的监控边界。

> **Note**
>
> 请注意以下事项：
>
> - 要通过互联网监测仪生成有意义的输出，您添加的 VPC 必须通过配置互联网网关连接到互联网。
> - 您可以添加 VPC 和 CloudFront 分配的组合，也可以添加 WorkSpaces 目录，还可以添加网络负载均衡器。您不能将网络负载均衡器或 WorkSpaces 目录与其他类型的资源一起添加。

6. 选择要监测的互联网流量百分比。
7. （可选）在高级设置下指定其他选项。

   - 对于城市网络最大限制，您可以选择网络监测仪将监测流量的城市网络（位置和 ASN 或互联网服务提供商）的数量限制。您可以通过编辑监测仪随时更改此设置。请参阅 选择一个城市-网络最大限制。

     要重置为默认值，请输入 `500000`。

     如果您设置了城市-网络最大限制，则无论您选择监测的流量百分比为多少，则该值将成为网络监测仪为您的应用程序监测的城市-网络数量上限。

   - （可选）您可以指定 Amazon S3 存储桶名称并自定义前缀，将所有受监测城市网络的互联网测量数据发布到 Amazon S3。

     网络监测仪会存储客户端位置和 ASN 对（即城市-网络）的互联网监测数据。网络监测仪还会为指向您的应用程序以及每个 AWS 区域和边缘站点的流量创建聚合 CloudWatch 指标。此外，网络监测仪还会每五分钟将排名前 500（按流量）的应用程序互联网监测数据发布到 CloudWatch Logs，以支持使用 CloudWatch 工具和其他方法来处理您的数据。如果您选择将测量数据发布到 S3，则测量数据仍会发布到 CloudWatch Logs。有关更多信息，请参阅 在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3。

   - （可选）您可以为监测仪添加标签。
8. 选择 Create monitor（创建监控）。

创建监测仪后，您可以随时编辑监测仪，例如更改应用程序流量百分比、更新城市-网络最大限制，或者添加或移除资源。您还可以删除监测仪。要在网络监测仪控制台中执行这些任务，请选择一个监测仪，然后在操作菜单中选择一个选项。但请注意，您不能更改监测仪的名称。

查看 Internet Monitor 控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择网络监控，然后选择互联网监视器。

   Monitors（监视器）选项卡显示了您创建的监视器列表。

要查看有关特定监测仪的更多信息，请选择该监测仪。

向监测仪添加资源

创建监测仪时，您必须将应用程序的资源与之关联：Amazon 虚拟私有云（VPC）、网络负载均衡器、Amazon CloudFront 分配、网络负载均衡器（NLB）或 Amazon WorkSpaces 目录。然后，网络监测仪会知道您的应用程序面向互联网的流量和客户端的位置，且它可以创建和维护流量配置文件，以确定要为您的监测仪发布的相关测量结果。

您可以在网络监测仪中将以下类型的资源作为“受监控的资源”添加到监测仪。请注意，网络监测仪不支持在一个监测仪中同时添加不同类型的资源。

- VPC：您在区域中添加的每个 VPC 都是受监测的资源。在您添加 VPC 时，网络监测仪会监控 VPC 中任何面向互联网的应用程序的流量，例如，托管在 Amazon EC2 实例上、网络负载均衡器背后或 AWS Fargate 容器中的应用程序。
- 网络负载均衡器：您添加的每个 NLB 都是受监测的资源。
- CloudFront 分配：您添加的每个 CloudFront 分配都是受监测的资源。
- WorkSpaces 目录：您在区域中添加的每个 WorkSpaces 目录都是受监测的资源。

当您监测 VPC 的流量时，会监测 VPC 背后的负载均衡器上托管的应用程序的流量。您可以选择监测单个网络负载均衡器负载均衡器的流量，而不是监测具有多个负载均衡器的 VPC。例如，如果您需要了解和配置功能以提高负载均衡器级别的性能或效率，这可能会很有帮助。或者，您可能需要网络负载均衡器级别的合规性信息。

在网络监测仪中向监测仪添加资源时，请注意以下几点：

- 要通过互联网监测仪生成有意义的输出，您添加的 VPC 必须通过配置互联网网关连接到互联网。
- 网络监测仪不支持在一个监测仪中同时添加不同类型的资源。

在将 VPC 或 NLB 添加为资源时，需要注意几个有关选择加入区域的区域差异。有关更多信息，请参阅 支持 Amazon CloudWatch 网络监测仪的 AWS 区域。

此外，测量最终用户端延迟的资源也存在差异。对于网络监测仪的延迟测量数据，VPC、NLB 和 WorkSpaces 目录不包含最终用户端的延迟。

选择要监测的应用程序流量百分比

您为要监测的应用程序流量百分比选择的覆盖范围，决定了要为您的应用程序监测的城市-网络（客户端位置和 ASN，后者通常是互联网服务提供商）数量，但最高不超过您可以同时设置的可选城市-网络最大限制。

如果您选择要监测的应用程序流量少于 100%，则监测仪可能存在可观测性误差。这是因为，如果Amazon CloudWatch网络监测仪创建了一些运行状况事件，但您没有监测这些流量，则您不会注意到这些问题。此外，有关客户端访问应用程序的性能和可用性分数信息的覆盖范围也可能较小。

以下各部分介绍了探索流量百分比设置和覆盖范围的选项，以及了解增加或减少覆盖范围之影响的选项。

- 探索更改应用程序流量百分比
- 查看在不同流量百分比设置下监测的城市网络数量

探索更改应用程序流量百分比的选项

您可以查看百分比更改时监测的城市-网络数量，从而探索如何将应用程序流量百分比更改为理想的值。这一部分详细说明了分步操作过程。

在网络监测仪控制台中，您可以尝试增加或减少监测仪的应用程序流量百分比，并查看调整后将覆盖的城市-网络估计数量。使用此选项，您可以快速查看流量百分比更改将如何影响监测的城市监测仪数量。这有助于您探索如何为应用程序选择适合的应用程序流量百分比。

探索监测覆盖范围并更新所监测流量的百分比

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 在监测仪列表中，选择一个监测仪。
4. 在配置选项卡的查看和评估流量覆盖范围部分中，您可以根据所选择的流量百分比评估对受监测城市-网络总数的影响。您还可以更新监测的流量百分比或更改监测仪的城市-网络数限制。

    - 探索流量百分比选项：在比较流量覆盖范围选项下的下拉菜单中，选择一个或多个流量百分比进行绘制和比较。对于您选择的每个流量百分比，您可以在设置该流量百分比覆盖范围时查看将要监测的城市-网络数量。

要了解更多信息，请参阅查看在不同百分比下监测的城市-网络数量。

- 更改监控覆盖范围：在了解其他流量覆盖选项下，选择更新监控覆盖范围。

  在浏览和设置流量监测覆盖范围对话框中，单击箭头以增加或减少要监测的流量百分比。如果选择 100% 流量，则可以看到监测的城市-网络总数，并全面监测您的应用程序流量。

  注意：要详细了解监测的城市-网络数量（此处为估计数）可能如何影响您的成本，请选择 CloudWatch 定价计算器链接，然后向下滚动到网络监测仪。

  要设置新的流量监测百分比，请选择更新监测覆盖范围。要保持当前的覆盖级别，请选择取消。

查看在不同流量百分比设置下监测的城市-网络数量

您可以查看在不同应用程序流量百分比下，将为您的应用程序监测的城市-网络数量。这一部分详细说明了分步操作过程。

您可以在网络监测仪控制台中查看相关的图表，这些图表将显示按照您指定的间隔时间，在不同的应用程序流量百分比下，城市-网络的覆盖范围将会如何变化。通过这种方法，您只需一张图表，即可快速可视化显示和比较特定流量百分比下的应用程序监测覆盖范围。

查看显示应用程序流量百分比和相应城市-网络覆盖范围的图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 在监测仪列表中，选择一个监测仪。
4. 选择流量洞察选项卡，然后向下滚动到网络流量图。
5. 在比较流量覆盖选项下的下拉列表中，选择一个或多个百分比。您可以选择一个或多个应用程序流量百分比，然后系统会更新监测的城市-网络总数图表，以显示网络监测仪为该流量百分比提供的监测覆盖范围。如果选择 100% 流量时的城市-网络数，您可以查看当全面监测您的应用程序流量时，将监测的城市-网络数量。

记住以下内容：

- 流量覆盖范围根据应用程序流量前一小时的城市网络数量计算得出。这意味着，在您选择要监测的特定流量百分比后，为您的应用程序监测的城市网络数量可能少于此处的流量覆盖范围比较图中显示的数量。

- 要确保监测所有应用程序流量，请将 `TrafficPercentageToMonitor` 设置为 100 而不设置 `MaxCityNetworksToMonitor`。或者，您可以将 `MaxCityNetworksToMonitor` 设置为 500000（网络监测仪中的上限）。
- 如果您设置了城市网络最大限制，则无论您选择哪个应用程序流量百分比选项，受监测的城市网络总数都不会超过该限制。
- 您可以详细了解受监测的城市-网络数量会如何影响您的成本。在 CloudWatch 定价计算器页面上，向下滚动到网络监测仪。

要设置新的流量监测百分比，请在探索其他流量覆盖范围选项下，选择更新监测覆盖范围。在此对话框中，选择一个流量百分比，然后选择更新监测覆盖范围。

选择一个城市-网络最大限制

Amazon CloudWatch 网络监测仪可以监测客户端访问应用程序资源的部分或所有位置的应用程序流量，以及客户端访问应用程序所使用的所有 ASN（通常是互联网服务提供商），即应用程序访问互联网所用的城市-网络。您需要在创建监测仪时选择要监测的应用程序流量百分比，然后您可以随时通过编辑监测仪来更新该值。

除了设置流量百分比外，您还可以设置监测的城市-网络数量上限值。这一部分说明了如何利用城市-网络限制来帮助管理账单成本，此外还提供了相关的信息和示例，以便您确定要设置的限制。

您设置的城市-网络数量上限值有助于确保账单可预测。有关更多信息，请参阅 Amazon CloudWatch 定价。您还可以使用 CloudWatch 价格计算器，来了解实际监测的城市-网络数量不同会对账单造成什么影响。要了解选项，请在 CloudWatch 定价计算器页面上向下滚动到 Internet Monitor。

要更新您的监测仪并更改城市网络最大限制，请参阅 编辑或删除网络监测仪。

如何利用城市-网络最大限制来控制账单

设置监测的城市-网络数量上限值，有利于防止账单中出现意外费用。例如，如果您的流量模式差异很大，这会非常有帮助。在监测的城市-网络数量超过 100 个后（每个账户的所有监测仪），每个额外的城市-网络都会增加账单成本。如果您设置了城市-网络最大限制，则无论您选择监测的流量百分比为多少，则该值将成为网络监测仪为您的应用程序监测的城市-网络数量上限。

您只需为实际监测的城市网络数量付费。您选择的城市网络最大限制允许在 Internet Monitor 使用监测仪监测流量时，设置可以包含的总数上限。您可以通过编辑监测仪随时更改最大限制。

要了解选项，请在 CloudWatch 定价计算器页面上向下滚动到 Internet Monitor。有关 Internet Monitor 定价的更多信息，请参阅 Amazon CloudWatch 定价页面上的 Internet Monitor 部分。

如何选择城市-网络最大限制

为便于您决定要选择的城市-网络最大限制，请考虑要监测的应用程序流量大小。创建监测仪后，下列 Internet Monitor 指标可以帮助您分析流量的使用情况和覆盖范围：CityNetworksMonitored、TrafficMonitoredPercent 以及一个或多个 CityNetworksFor*NN*PercentTraffic 指标，其中 *NN* 是以下值之一的百分比值：25、50、90、95、99 或 100。要查看这些指标以及所有其他网络监测仪指标的定义，请参阅 将 CloudWatch Metrics 与 Amazon CloudWatch 网络监测仪结合使用。

要查看网络流量覆盖范围概览图，请转到 CloudWatch 控制面板上的流量洞察选项卡，然后在网络流量图部分中，为流量覆盖范围比较选项选择一个选项。这一部分中的图表显示了实际为应用程序监测的城市-网络数量，以及您在下拉列表中选择的不同应用程序流量百分比的图表线。要了解更多信息，请参阅 设置应用程序流量百分比。

要更详细地了解您的选项，您可以使用 Internet Monitor 指标，如以下示例所述。这些示例说明了如何根据所需的应用程序互联网流量覆盖范围来选择最适合您的城市网络最大限制。在 CloudWatch Metrics 中使用 Internet Monitor 指标进行查询可以帮助您更详细地了解应用程序互联网流量的覆盖范围。

确定城市-网络最大限制的示例

例如，假设您设置了 100 个城市网络的最大监测限制，并且应用程序由 2637 个城市网络的客户端访问。在 CloudWatch Metrics 中，您会看到返回的以下 Internet Monitor 指标：

```
CityNetworksMonitored 100
TrafficMonitoredPercent  12.5
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

在此示例中，您可以看到目前正在监测 12.5% 的互联网流量，最大限制设置为 100 个城市网络。如果您想监测 90% 的流量，则下一个指标将提供相关信息：CityNetworksFor90PercentTraffic 表示您需要监测 2143 个城市网络才能实现 90% 的覆盖率。为此，您需要更新监测仪并将城市网络最大限制设置为 2143。

同样，假设您想监测应用程序 100% 的互联网流量。为此，下一个指标 CityNetworksFor100PercentTraffic 表示，要实现 100% 的监测，您应更新监测仪，并城市网络最大限制设置为 2637。

如果您现在将最大限制设置为 5000 个城市网络，由于该值大于 2637，您将看到返回的以下指标：

```
CityNetworksMonitored 2637
TrafficMonitoredPercent  100
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

从这些指标中您可以看出，限制越高，您就可以监测所有 2637 个城市网络，占互联网流量的 100%。

在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3

您可以选择让 Amazon CloudWatch 网络监测仪将互联网监测数据发布到 Amazon S3，以便将面向互联网的流量发送到监测仪中受监测的城市-网络（客户端位置和 ASN，通常是互联网服务提供商），最多不能超过 50 万个城市-网络的服务限制。Internet Monitor 每五分钟自动将每个监测仪的前 500 个（按流量计）城市网络的互联网测量数据发布到 CloudWatch Logs。发布到 S3 的测量数据包括发布到 CloudWatch Logs 的前 500 个测量数据。

您可以选择发布到 S3 的选项，并指定在创建或更新监测仪时要将测量数据发布到的存储桶。必须先在 S3 中创建存储桶，然后才能在 Internet Monitor 中指定存储桶。发布到 S3 的互联网测量数据的服务限制为 50 万个城市网络。Internet Monitor 将互联网测量数据作为事件发布到 S3，事件是存储在存储桶中的一系列压缩日志文件对象。

您为 Internet Monitor 创建要向其中发布测量数据的 S3 存储桶时，请确保遵守 CloudWatch Logs 提供的权限指南。这样做可以确保 Internet Monitor 可以直接将日志发布到 S3，并且 AWS 可以在需要时创建和更改与接收日志的日志组关联的资源策略。有关更多信息，请参阅《Amazon CloudWatch Logs 用户指南》中的 发送到 CloudWatch Logs 的日志。

发布的日志文件是压缩文件。如果您使用 Amazon S3 控制台打开这些日志文件，则将对其进行解压缩，并且将显示互联网测量事件。如果您下载这些文件，则必须对其进行解压缩才能查看事件。

您还可以使用 Amazon Athena 查询日志文件中的互联网测量数据。Amazon Athena 是一种交互式查询服务，让您能够更轻松地使用标准 SQL 分析 Amazon S3 中的数据。有关更多信息，请参阅 使用 Amazon Athena 查询 Amazon S3 日志中的互联网测量数据。

使用 Internet Monitor 监测仪

创建 Amazon CloudWatch 网络监测仪后，有多种使用方法：例如，您可以在 CloudWatch 控制面板中查看信息、使用 AWS Command Line Interface 获取信息以及设置运行状况警报。

监测仪提供了有关应用程序和配置首选项的信息，以便网络监测仪可以为您自定义要在事件中发布的测量数据和指标。Internet Monitor 从 AWS 的全球基础设施足迹中收集测量数据。这些测量数据来自世界各地的大量网络性能和可用性信息。通过使用您为应用程序添加的资源的信息，Internet Monitor 会

为您发布性能和可用性测量数据，这些数据的范围仅限于应用程序处于活动状态的城市网络（即客户端位置和 ASN，通常是互联网服务提供商或 ISP）。因此，Internet Monitor 控制面板和 CloudWatch Logs 中的测量数据和性能（关于可用性、性能、传输的监测字节和往返时间）特定于您的客户端位置和 ASN。

网络监测仪还会确定性能和可用性何时出现异常。默认情况下，Internet Monitor 会将您的流量与 AWS 为您所在客户端位置的每个源/目标对收集的可用性和性能测量数据进行叠加，以确定性能或可用性何时出现明显下降。当应用程序的位置和范围出现显著下降时，Internet Monitor 会生成运行状况事件，并将有关该问题的信息发布到您的监测仪。

创建监测仪后，您可以通过以下方式，使用监测仪访问网络监测仪提供的信息或接收相关提醒：

- 使用 CloudWatch 控制面板查看和了解性能、可用性和运行状况事件，浏览应用程序的历史数据，并深入了解配置应用程序提高性能的新方法。要了解更多信息，请参阅以下内容：
  - 在网络监测仪中跟踪实时性能和可用性（“概览”页面）
  - 在网络监测仪中分析历史数据（“分析”页面）
  - 在网络监测仪中获取优化应用程序性能的建议（“优化”页面）
- 配置运行状况事件阈值可更改触发网络监测仪为您的应用程序创建运行状况事件的因素。您可以配置总体阈值和本地（城市网络）阈值。要了解更多信息，请参阅更改运行状况事件阈值。
- 将 AWS CLI 命令与网络监测仪 API 操作结合使用，以查看流量配置文件信息、查看测量数据、列出运行状况事件等。要了解更多信息，请参阅 将 CLI 与 Amazon CloudWatch 网络监测仪结合使用的示例。
- 使用标准 CloudWatch 工具（如 CloudWatch Contributor Insights、CloudWatch Metrics Explorer 和 CloudWatch Logs Insights）来可视化 CloudWatch 中的数据。要了解更多信息，请参阅 使用 CloudWatch 工具和网络监测仪查询接口探索您的数据。
- 如果您启用了将测量结果发布到 S3 的功能，则结合使用 Athena 和 S3 日志来访问和分析应用程序的网络监测仪互联网测量结果。
- 创建 Amazon EventBridge 通知，以便在网络监测仪确定有运行状况事件时提醒您。要了解更多信息，请参阅 将 Amazon CloudWatch 网络监测仪与 Amazon EventBridge 结合使用。
- 在网络监测仪确定问题是由 AWS 网络引起时，自动接收 AWS Health Dashboard 通知。该通知包括 AWS 为缓解问题而采取的步骤。

编辑或删除网络监测仪

使用操作菜单，您可以在创建监测仪后在 Amazon CloudWatch 网络监测仪中编辑或删除监测仪。例如，您可以通过编辑监测仪来执行以下操作：

- 更改应用程序流量监测百分比
- 设置或更新城市-网络最大限制
- 更改运行状况事件的可用性或性能分数阈值
- 添加或移除资源类型
- 启用或更新向 Amazon S3 发布的事件

您还可以删除监测仪。请注意，监测仪创建后将不能更改名称。

要更改监视器或删除监视器，请按照以下过程中的一种进行操作。

编辑监测仪

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 选择您的监测仪，然后选择操作菜单。
4. 选择更新监测仪。
5. 进行所需的更新。例如，要更改流量监测百分比，请在要监测的应用程序流量下，选择或输入一个百分比。
6. 选择更新。

删除监视器

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 选择您的监测仪，然后选择操作菜单。
4. 选择 禁用。
5. 再次选择操作菜单，然后选择删除。

要详细了解您可以更新的选项，请参阅以下内容：

- 要详细了解您可以在网络监测仪中添加的资源，请参阅 向监测仪添加资源。
- 要详细了解应用程序流量百分比，请参阅 选择要监测的应用程序流量百分比。
- 要了解有关更改运行状况事件阈值的更多信息，请参阅更改运行状况事件阈值。
- 要详细了解城市-网络最大限制，请参阅 选择一个城市-网络最大限制。

- 要详细了解有关将事件发布到 S3 的选项，请参阅 在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3。

更改网络监测仪的运行状况事件阈值

您可以围绕网络监测仪为应用程序创建运行状况事件的方式和时间配置多个选项。要进行更改，请访问网络监测仪控制台中的配置页面。

您可以更改将会触发网络监测仪创建运行状况事件的总体阈值。默认的运行状况事件性能和可用性阈值均为 95%。也就是说，当应用程序的总体性能或可用性分数降至 95% 或以下时，网络监测仪会创建运行状况事件。对于总体阈值，运行状况事件可以由单个较大的问题触发，也可以由多个较小的问题组合触发。

您还可以更改本地（即城市网络）阈值，该阈值如果与总体影响级别的百分比结合起来，将触发运行状况事件。例如，通过设置一个阈值，使其在一个或多个城市网络（地点和 ASN，通常是 ISP）的分数降至阈值以下时创建运行状况健康事件，您可以深入了解流量较低的地点何时出现问题。

一个额外的本地阈值选项可与本地阈值配合使用，用于可用性或性能分数。第二个因素是在网络监测仪根据本地阈值创建运行状况事件之前必然受到影响的流量占整体流量的百分比。

通过为整体流量和本地流量配置阈值选项，您可以微调创建运行状况事件的频率，以符合您的应用程序使用量和需求。请注意，如果您将本地阈值设置得较低，通常会创建较多的运行状况事件，具体取决于您的应用程序和您设置的其他阈值配置值。

总之，您可以通过以下方式为性能分数、可用性分数或两者配置运行状况事件阈值：

- 为触发运行状况事件选择不同的全局阈值。
- 为触发运行状况事件选择不同的本地阈值。使用此选项，您还可以更改网络监测仪创建事件之前必须超过的对整个应用程序的影响百分比。
- 选择关闭基于本地阈值触发运行状况事件，或启用本地阈值选项。

您还可以配置有关性能分数、可用性分数或两者的选项。您可以配置这些选项的组合，也可以只配置其中一个。

要更新性能分数和/或可用性分数的阈值和其他配置选项，请执行以下操作：

更改阈值配置选项

1. 在AWS Management Console中，导航到 CloudWatch，然后在左侧导航窗格中选择“网络监测仪”。

2. 进入控制台后，在监测仪的任意页面上，选择编辑监测仪部分，然后选择更新阈值。
3. 在打开的对话框页面上，为触发网络监测仪创建运行状况事件的阈值和其他选项选择所需的新值和选项。您可以执行以下任意操作：

   - 为可用性分数阈值、性能分数阈值，或同时为两者选择一个新值。

     每个设置部分中的图表将显示应用程序可用性或性能的当前阈值设置和最近的实际运行状况事件分数。通过查看典型值，您可以了解可能需要将阈值更改为哪些值。

     提示：要查看大图并更改时间范围，请选择图表右上角的展开标志。
   - 选择开启或关闭有关可用性阈值或性能或两者的本地阈值。启用某个选项后，可以设置希望网络监测仪创建运行状况事件时的阈值和影响级别。
4. 配置阈值选项后，选择更新运行状况事件阈值以保存更新。

要详细了解运行状况事件的工作原理，请参阅网络监测仪何时会创建和解决运行状况事件。

使用 Amazon VPC 添加或创建 Amazon CloudWatch 网络监测仪

在 AWS Management Console 中创建 Amazon Virtual Private Cloud VPC 时，您可以选择在 Amazon CloudWatch 网络监测仪中为其设置监测。您可以将 VPC 添加到现有监测仪中，也可以选择在 Amazon VPC 控制台中为该 VPC 创建新的监测仪。

通过将网络监测仪与 VPC 结合使用，您可以查看和评估特定于应用程序客户端位置和 ASN（通常是互联网服务提供商）的测量数据和指标，这些测量数据和指标涉及可用性、性能、传输的监测字节数和往返时间。网络监测仪还会确定性能和可用性何时出现异常，并在监测仪中创建运行状况事件，您可以选择是否接收相关通知。要详细了解如何使用监测仪来管理和改善客户使用应用程序的体验，请参阅 使用 Internet Monitor 监测仪。

> **Important**
>
> 您必须拥有正确的权限，才能创建监测仪或向现有监测仪添加 VPC。有关更多信息，请参阅适用于 Amazon CloudWatch 网络监测仪的 Identity and Access Management。

向现有监测仪添加 VPC

当您在 AWS Management Console 中创建 VPC 时，可以选择让 Amazon CloudWatch 网络监测仪为现有监测仪添加新 VPC。添加 VPC 后，请等待几分钟，网络监测仪控制台才会开始显示该 VPC 的指标。

您可以随时编辑监测仪，以便删除 VPC 或添加其他 VPC 或其他资源。您还可以更改正在监测的流量百分比，也可以进行其他更改。如果您选择从监测仪中删除 VPC，网络监测仪将不再监测从客户端到该 VPC 的流量。

要了解有关更新监测仪的更多信息，请参阅 编辑或删除网络监测仪。

为 VPC 创建监测仪

如果您选择为 VPC 创建监测仪，则创建监测仪向导将引导您完成相应步骤。在创建监测仪时，将 VPC 添加为受监测资源。如有需要，还可以针对应用程序选择要监测的客户端流量百分比（默认值为 100%）。

您可以通过查看 使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪 中的信息来了解更多信息。

定价

使用 Amazon CloudWatch 网络监测仪只需按实际用量付费。Internet Monitor 的定价包含两个部分：按监测的资源收费和按城市网络收费。城市网络是客户端借以访问应用程序资源的位置，也是客户端访问资源所使用的网络（ASN，例如互联网服务提供商或 ISP）。

包括定价示例在内的有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的定价

停止监测 VPC

如果不想继续使用网络监测仪来监测 VPC 资源，请在网络监测仪控制台中执行以下操作：

从监测仪中删除资源

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 选择您的监测仪，然后选择操作菜单。
4. 选择更新监测仪。
5. 在已添加的资源下，选择删除资源。
6. 选择要删除的 VPC，然后选择删除。
7. 选择更新。

通过 CloudFront 添加或创建 Amazon CloudWatch 网络监测仪

在 Amazon CloudFront 控制台中的分配的指标控制面板上，您可以在 Amazon CloudWatch 网络监测仪中为分配设置其他监测。您可以将分配添加到现有监测仪中，也可以为该分配创建新的监测仪。

将网络监测仪与 CloudFront 分配结合使用后，便可以查看和评估特定于您的应用程序客户端位置和 ASN（通常是互联网服务提供商）的测量数据和指标，这些测量数据和指标涉及可用性、性能、传输的监测字节数和往返时间。网络监测仪还会确定性能和可用性何时出现异常，并在监测仪中创建运行状况事件，您可以选择是否接收相关通知。要详细了解如何使用监测仪来管理和改善客户使用应用程序的体验，请参阅 使用 Internet Monitor 监测仪。

> **Important**
>
> 您必须拥有正确的权限，才能创建监测仪或向监测仪添加分配。有关更多信息，请参阅 适用于 Amazon CloudWatch 网络监测仪的 Identity and Access Management。

向现有监测仪添加分配

您可以选择让网络监测仪直接从 AWS Management Console 中的 CloudFront 指标控制面板向现有监测仪添加分配。添加分配后，请等待几分钟，网络监测仪控制台才会开始显示该分配的指标。

您可以随时编辑监测仪，以删除分配或添加其他分配或其他资源。您还可以更改正在监测的流量百分比，也可以进行其他更改。如果您选择从监测仪中删除分配，网络监测仪将不再监测从客户端到该分配的流量。

要了解有关更新监测仪的更多信息，请参阅 编辑或删除网络监测仪。

为分配创建监测仪

如果您选择为分配创建监测仪，则创建监测仪向导将引导您完成相应步骤。在创建监测仪时，将分配添加为受监测资源。如有需要，还可以针对应用程序选择要监测的客户端流量百分比（默认值为 100%）。

您可以通过查看 使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪 中的信息来了解更多信息。

定价

使用 Amazon CloudWatch 网络监测仪只需按实际用量付费。Internet Monitor 的定价包含两个部分：按监测的资源收费和按城市网络收费。城市网络是客户端借以访问应用程序资源的位置，也是客户端访问资源所使用的网络（ASN，例如互联网服务提供商或 ISP）。

包括定价示例在内的有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的定价

停止监测分配

如果不想继续使用网络监测仪来监测分配资源，请在网络监测仪中执行以下操作：

从监测仪中删除资源

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中的网络监控下，选择互联网监视器。
3. 选择您的监测仪，然后选择操作菜单。
4. 选择更新监测仪。
5. 在已添加的资源下，选择删除资源。
6. 选择要删除的分配，然后选择删除。
7. 选择更新。

将 CLI 与 Amazon CloudWatch 网络监测仪结合使用的示例

本节包括将 AWS Command Line Interface 与 Amazon CloudWatch 网络监测仪操作结合使用的示例。

在开始之前，请确保您登录以将 AWS CLI 与相同 AWS 账户结合使用，该账户具有要监控的 Amazon 虚拟私有云（VPC）、网络负载均衡器、Amazon CloudFront 分配或 Amazon WorkSpaces 目录。Internet Monitor 不支持跨账户访问资源。有关使用 AWS CLI 的更多信息，请参阅 AWS CLI 命令参考。有关将 API 操作与 Amazon CloudWatch 网络监测仪结合使用的更多信息，请参阅 Amazon CloudWatch 网络监测仪 API 参考指南。

主题

- 创建监视器
- 查看显示器详细信息
- 列出运行状况事件
- 查看特定的运行状况事件
- 查看监视器列表
- 编辑监视器
- 删除监视器

创建监视器

在 Internet Monitor 中创建监视器时，您需要提供一个名称并将资源与监视器相关联，以显示应用程序的互联网流量。您可以指定流量百分比，该百分比定义了您的应用程序流量受到监测的程度。这也决定了受到监测的城市网络（即客户端位置和 ASN，通常是互联网服务提供商或 ISP）的数量。您还可以为要监测的应用程序资源的城市网络最大数量设置限制，以帮助您控制账单成本。有关更多信息，请参阅 选择一个城市-网络最大限制。

最后，您可以选择是否要将应用程序的所有互联网测量结果发布到 Amazon S3。网络监测仪会将前 500 个（按流量计）城市网络的互联网测量数据自动发布到 CloudWatch Logs，但您可以选择同时将所有测量结果发布到 S3。

要使用 AWS CLI 创建监测仪，请使用 `create-monitor` 命令。以下命令将创建一个监测仪，该监测仪可监控 100% 的流量，但将城市网络的最大限制设置为 10,000，添加一个 VPC 资源，并选择将互联网测量结果发布到 Amazon S3。

> **Note**
>
> Internet Monitor 每五分钟将向每个监测仪发送流量的前 500 个城市网络（客户端位置和 ASN，通常是互联网服务提供商或 ISP）的互联网测量数据发布到 CloudWatch Logs。或者，您可以选择将所有受监测城市网络（最多 50 万个城市网络的服务限制）的互联网测量数据发布到 Amazon S3 存储桶。有关更多信息，请参阅 在 Amazon CloudWatch 网络监测仪中将互联网测量数据发布到 Amazon S3。

```
aws internetmonitor --create-monitor monitor-name "TestMonitor" \
    --traffic-percentage-to-monitor 100 \
    --max-city-networks-to-monitor 10000 \
    --resources "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --internet-measurements-log-delivery
 S3Config="{BucketName=MyS3Bucket,LogDeliveryStatus=ENABLED}"
```

```
{
"Arn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
 "Status": "ACTIVE"
}
```

> **Note**
>
> 您不能更改监测仪的名称。

查看显示器详细信息

要使用 AWS CLI 查看有关监测仪的信息，请使用 `get-monitor` 命令。

```
aws internetmonitor get-monitor --monitor-name "TestMonitor"
```

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
}
```

列出运行状况事件

当应用程序的互联网流量性能下降时，Internet Monitor 会在监视器中创建运行状况事件。要使用 AWS CLI 查看当前运行状况事件列表，请使用 `list-health-events` 命令

```
aws internetmonitor list-health-events --monitor-name "TestMonitor"
```

```
{
    "HealthEvents": [
        {
            "EventId": "2022-06-20T01-05-05Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:15:14Z",
            "ServiceLocations": [
                {
```

```
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.21,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 60.370000000000005,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                }
            ],
            "StartedAt": "2022-06-20T01:05:05Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-05-05Z/latency"
        },
        {
            "EventId": "2022-06-20T01-17-56Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:30:23Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.29,
            "ClientLocations": [
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 75.32,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
```

```
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 22.91,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:17:56Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-17-56Z/latency"
        },
        {
            "EventId": "2022-06-20T01-34-20Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:35:04Z",
            "ServiceLocations": [
                {
```

```
                "Name": "us-east-1"
            }
        ],
        "PercentOfTotalTrafficImpacted": 1.15,
        "ClientLocations": [
            {
                "City": "Lockport",
                "PercentOfClientLocationImpacted": 39.45,
                "PercentOfTotalTraffic": 2.01,
                "Country": "United States",
                "Longitude": -78.6913,
                "AutonomousSystemNumber": 26101,
                "Latitude": 43.1721,
                "Subdivision": "New York",
                "NetworkName": "YAHOO-BF1"
            },
            {
                "City": "Toronto",
                "PercentOfClientLocationImpacted": 29.770000000000003,
                "PercentOfTotalTraffic": 1.05,
                "Country": "Canada",
                "Longitude": -79.3623,
                "AutonomousSystemNumber": 14061,
                "Latitude": 43.6547,
                "Subdivision": "Ontario",
                "CausedBy": {
                    "Status": "ACTIVE",
                    "Networks": [
                        {
                            "AutonomousSystemNumber": 16509,
                            "NetworkName": "Amazon.com"
                        }
                    ],
                    "NetworkEventType": "AWS"
                },
                "NetworkName": "DIGITALOCEAN-ASN"
            },
            {
                "City": "Hangzhou",
                "PercentOfClientLocationImpacted": 2.88,
                "PercentOfTotalTraffic": 0.7799999999999999,
                "Country": "China",
                "Longitude": 120.1612,
                "AutonomousSystemNumber": 37963,
```

```
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:34:20Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-34-20Z/latency"
        }
    ]
}
```

查看特定的运行状况事件

要使用 CLI 查看有关特定运行状况事件的更多详细信息，请使用您的监视器名称和运行状况事件 ID 运行 get-health-event 命令。

```
aws internetmonitor get-monitor --monitor-name "TestMonitor" --event-id "health-event/
TestMonitor/2021-06-03T01:02:03Z/latency"
```

```
{
    "EventId": "2022-06-20T01-34-20Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:35:04Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor/
health-event/2022-06-20T01-34-20Z/latency",
    "LastUpdatedAt": "2022-06-20T01:35:04Z",
    "ClientLocations": [
        {
            "City": "Lockport",
            "PercentOfClientLocationImpacted": 39.45,
            "PercentOfTotalTraffic": 2.01,
            "Country": "United States",
            "Longitude": -78.6913,
            "AutonomousSystemNumber": 26101,
            "Latitude": 43.1721,
            "Subdivision": "New York",
```

```
            "NetworkName": "YAHOO-BF1"
        },
        {
            "City": "Toronto",
            "PercentOfClientLocationImpacted": 29.770000000000003,
            "PercentOfTotalTraffic": 1.05,
            "Country": "Canada",
            "Longitude": -79.3623,
            "AutonomousSystemNumber": 14061,
            "Latitude": 43.6547,
            "Subdivision": "Ontario",
            "CausedBy": {
                "Status": "ACTIVE",
                "Networks": [
                    {
                        "AutonomousSystemNumber": 16509,
                        "NetworkName": "Amazon.com"
                    }
                ],
                "NetworkEventType": "AWS"
            },
            "NetworkName": "DIGITALOCEAN-ASN"
        },
        {
            "City": "Shenzhen",
            "PercentOfClientLocationImpacted": 4.07,
            "PercentOfTotalTraffic": 0.61,
            "Country": "China",
            "Longitude": 114.0683,
            "AutonomousSystemNumber": 37963,
            "Latitude": 22.5455,
            "Subdivision": "Guangdong",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        },
        {
            "City": "Hangzhou",
            "PercentOfClientLocationImpacted": 2.88,
            "PercentOfTotalTraffic": 0.7799999999999999,
            "Country": "China",
            "Longitude": 120.1612,
            "AutonomousSystemNumber": 37963,
            "Latitude": 30.2994,
            "Subdivision": "Zhejiang",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
```

```
        }
    ],
    "StartedAt": "2022-06-20T01:34:20Z",
    "ImpactType": "PERFORMANCE",
    "PercentOfTotalTrafficImpacted": 1.15
}
```

查看监视器列表

要使用 CLI 查看您账户中所有监视器的列表，请运行 `list-monitors` 命令。

```
aws internetmonitor list-monitors
```

```
{
    "Monitors": [
        {
            "MonitorName": "TestMonitor",
            "ProcessingStatus": "OK",
            "Status": "ACTIVE"
        }
    ],
    "NextToken": " zase12"
}
```

编辑监视器

要使用 CLI 更新有关监视器的信息，请使用 `update-monitor` 命令并指定要更新的监视器的名称。您可以更新要监测的流量百分比、要监测的城市-网络最大数量限制、添加或移除网络监测仪用于监测流量的资源，以及将监测仪状态从 ACTIVE 改为 INACTIVE，反之亦然。请注意，您不能更改监视器的名称。

`update-monitor` 调用的响应仅返回 `MonitorArn` 和 `Status`。

以下示例显示了如何使用 `update-monitor` 命令将要监测的城市网络最大数量更改为 `50000`：

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --max-city-networks-to-
monitor 50000
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": " ACTIVE "
```

```
}
```

以下示例显示了如何添加和删除资源：

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" \
    --resources-to-add "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --resources-to-remove "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-2222444455556666"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
}
```

以下示例显示了如何使用 update-monitor 命令将监视器状态更改为 INACTIVE：

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --status "INACTIVE"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "INACTIVE"
}
```

删除监视器

您可以使用 delete-monitor 命令通过 CLI 删除监视器。首先，必须将监视器设置为非活动状态。为此，请使用 update-monitor 命令将状态更改为 INACTIVE。使用 get-monitor 命令并检查状态，确认监视器处于非活动状态。

当监视器状态为 INACTIVE 时，您可以使用 CLI 运行 delete-monitor 命令来删除监视器。成功的 delete-monitor 调用的响应为空。

```
aws internetmonitor delete-monitor --monitor-name "TestMonitor"
```

```
{}
```

使用 Internet Monitor 控制面板进行监控和优化

使用 AWS Management Console中的 Amazon CloudWatch 网络监测仪控制面板，您可以直观地显示数据，获取有关 AWS 应用程序互联网流量的见解和建议，以及配置监测仪的选项。

在创建监测仪来监测应用程序的互联网性能和可用性后，网络监测仪会存储您的客户端位置和 ASN 对（即城市-网络）的互联网监测数据。网络监测仪还会为指向您的应用程序以及每个 AWS 区域和边缘站点的流量创建聚合 CloudWatch 指标。您可以通过多种不同的方式筛选、探索来自监测仪的信息并从中获取可指导操作的建议。网络监测仪控制面板将引导您查看监测流量数据，并获取有关这些数据的见解。

首先，在 CloudWatch 控制台的网络监控下，选择互联网监视器。

> **Note**
>
> 这一部分主要介绍如何使用 AWS Management Console来筛选和查看网络监测仪指标。您还可以通过 AWS CLI 或 SDK 使用网络监测仪 API 操作，从而直接与存储在 CloudWatch 日志文件中的网络监测仪事件结合使用。有关更多信息，请参阅 使用监测仪和测量信息。有关 API 操作的更多信息，请参阅 将 CLI 与 Amazon CloudWatch 网络监测仪结合使用的示例 和 Amazon CloudWatch 网络监测仪 API 参考。

网络监测仪控制面板中有五个页面（选项卡）：

- 在概览页面上，您可以从整体上了解所监测的流量，包括当前性能和可用性信息、最近和当前运行状况事件的摘要，以及潜在可以提高客户端性能的主要建议。
- 在运行状况事件页面上，您可以查看当前正在影响或以前曾经影响客户端访问应用程序的位置的当前和历史运行状况事件。
- 在分析页面上，您可以查看受监测流量靠前的客户端位置（按流量）信息，这些信息将以多种可自定义的方式汇总。您还可以查看运行状况分数和指标的历史趋势。您可以按位置、ASN、日期等条件进行筛选，并直观地显示互联网流量指标随时间推移的变化。
- 在优化页面上，网络监测仪会根据流量模式和历史性能预测应用程序在主要 AWS 区域（或 Amazon CloudFront）的性能改进。对于每项主要配置，相关表格将按客户端位置提供延迟减少情况的明细。在第二页上，您可以选择多个区域（如果您愿意，还可包括 CloudFront 配置）来比较延迟减少情况。对于您选择的每个配置（区域），该页面都会显示按城市位置列出的相关延迟详情表。
- 在配置页面上，您可以查看监测仪详细信息并配置选项，例如要监测的流量百分比。

除了这些控制面板选项外，您还可以使用这些工具更深入地了解网络监测仪利用监视器收集的指标的详细信息。网络监测仪会生成并发布包含流量测量数据的日志文件，因此您可以在控制台中使用其他 CloudWatch 工具进一步可视化网络监测仪发布的数据，包括 CloudWatch Contributor Insights、CloudWatch Metrics 和 CloudWatch Logs Insights 等。有关更多信息，请参阅 使用 CloudWatch 工具和网络监测仪查询接口探索您的数据。

在以下部分中了解如何使用 Internet Monitor 来查看性能和可用性测量值。

主题

- 在网络监测仪中跟踪实时性能和可用性（“概览”页面）
- 在网络监测仪中查看运行状况事件和指标（“运行状况事件“页面）
- 在网络监测仪中分析历史数据（“分析”页面）
- 在网络监测仪中获取优化应用程序性能的建议（“优化”页面）
- 在网络监测仪中监测详细信息（“配置”页面）

在网络监测仪中跟踪实时性能和可用性（“概览”页面）

Amazon CloudWatch 网络监测仪控制台中的概览页面将简要展示监测仪所跟踪流量的性能和可用性，以及有关运行状况事件何时影响受监测流量的时间线。该页面还提供了有关配置更改的主要建议，这些建议有助于为在主要客户端位置（按流量）使用应用程序的客户端减少延迟。

流量概览和状态

流量概览部分显示了应用程序的整体可用性和性能。请注意，这一部分显示的是汇总的总体性能和可用性分数，考虑了应用程序指向所有最终用户和服务位置的所有流量。通过在分析选项卡上搜索和筛选监测信息，您可以查看特定客户端位置和服务位置的运行状况分数。

在状态下，您可以查看监测仪是在主动创建监测仪的数据，还是正在等待数据可用。您还可以查看正在监测的应用程序流量百分比。如果要更改百分比，请访问配置页面。

网络监测仪使用统计过程来创建受监测流量的可用性和性能分数。对于不同 ASN 和 AWS 服务在不同地理位置之间的网络流量，AWS 拥有大量关于互联网性能和可用性的历史数据。网络监测仪使用的连接数据由 AWS 从其全球网络中获取，用来计算互联网流量的性能和可用性基准。这与我们在 AWS 用来监控我们自己的互联网正常运行时间和可用性的数据相同。

以这些测量值为基准，Internet Monitor 可以检测到应用程序的互联网性能和可用性与基准相比何时有所下降。为了更容易看到下降情况，我们以性能分数和可用性分数的形式向您报告此信息。

有关更多信息，请参阅 AWS 如何计算性能和可用性分数。

运行状况事件时间线

运行状况事件时间线图会显示过去 24 小时内发生的运行状况事件。图形下方的摘要显示应用程序当前和最近受到的影响。要获取详细信息，您可以选择查看更多运行状况事件。

要更改运行状况事件的阈值，请转到配置页面。

减少主要区域的延迟

网络监测仪会自动评估您当前应用程序配置使用最多的 AWS 区域（即客户端数量最高的区域），并确定是否有其他区域能够为客户端提供更好的总第一字节时间（TTFB）。

请注意，由于这是汇总的 TTFB，因此如果将流量从一个区域转移到另一个区域，预计大多数位置的 TTFB 会有所改善，但某些区域的客户可能不会看到任何变化，甚至性能会降低。

要探索更多延迟改善建议，包括更精细的细节（例如按客户位置划分），请访问优化页面。

在网络监测仪中查看运行状况事件和指标（“运行状况事件“页面）

Amazon CloudWatch 网络监测仪控制台中的运行状况事件页面以地图的方式显示了影响应用程序客户端位置和 ASN 的运行状况事件。您可以单击地图上的圆圈以了解有关事件的更多详细信息。运行状况事件表列出了受事件影响的位置以及有关影响的详细信息。

互联网流量概览

互联网流量概览图显示了客户端访问应用程序的位置和 ASN 特有的互联网流量和运行状况事件。图上显示为灰色的国家/地区是包含应用程序流量的国家/地区。

图上的每个圆圈都表示在您选择的时间段内某个区域发生的运行状况事件。检测到 AWS 中托管的任一资源与客户端正在用于访问应用程序的城市-网络之间出现连接问题（即达到某个可自定义的特定阈值）时，网络监测仪会创建运行状况事件。

选择图上的圆圈将会显示有关该位置所发生运行状况事件的更多详细信息。此外，对于出现运行状况事件的集群，您可以在此图下方的 Health events（运行状况事件）表中查看详细信息。

请注意，如果确定某个事件对您的应用程序具有重大影响时，网络监测仪会在监测仪中创建运行状况事件。如果在您选择的时间段内，没有任何运行状况事件对客户端位置的流量造成超出阈值的影响，则此图为空白。有关更多信息，请参阅网络监测仪何时会创建和解决运行状况事件。

运行状况事件

运行状况事件表列出了受运行状况事件影响的客户端位置以及事件相关信息。表中包含以下列。

列	描述
事件类型	指示当前事件是整体运行状况事件还是局部运行状况事件，或者运行状况事件是否是历史事件。

列	描述
客户端位置	受事件影响、延迟增加或可用性降低的最终用户的位置。 要详细了解网络监测仪中的客户端位置准确性，请参阅 网络监测仪中的地理位置信息和准确性。
ISP 名称（ASN）	流量经过的网络。通常是网络流量的互联网服务提供商（ISP）或自治系统号（ASN）。
服务位置	网络流量的 AWS 位置，可以是 AWS 区域 或互联网边缘站点。
流量影响	该事件在延迟增加或可用性降低方面造成了多大影响。对于延迟，这是与使用此客户端网络从该客户端位置到该 AWS 位置的流量的典型性能相比，事件期间延迟增加的百分比。
影响类型	运行状况事件的影响类型。运行状况事件通常由延迟增加（性能问题）或可到达性（可用性问题）引起。 您也可以单击影响类型来查看损害的原因。网络监测仪会尽量分析运行状况事件的起源，以确定该事件是由 AWS 还是 ASN（互联网服务提供商）引起的。 请注意，在事件解决后将继续此分析。Internet Monitor 可以在长达一个小时的时间内使用新信息更新事件。

如果您选择运行状况事件表中的客户端位置之一，则可以查看该位置的运行状况事件的更多详细信息。例如，您可以查看该事件何时开始、何时结束以及对本地流量的影响。

网络路径可视化

如果网络监测仪已完成事件的影响分析，则可以通过查看网络路径可视化来查看流向客户端位置的流量的完整网络路径。对于此客户端/位置对，该完整路径将显示应用程序在 AWS 位置和客户端位置之间的运行状况事件网络路径上的每个节点。

网络监测仪确定了受影响的原因后，网络监测仪会在节点周围添加一个红色虚线圆圈。受损可能是由 ASN [通常是互联网服务提供商（ISP）] 造成的，或者原因可能是 AWS。如果有多种原因造成受损，则系统会圈出多个节点。

在网络监测仪中分析历史数据（“分析”页面）

在 Amazon CloudWatch 网络监测仪控制台的分析页面上，您可以查看应用程序受监测流量排名靠前的主要客户端位置（按流量）。您还可以查看随时间变化的流量性能和可用性分数图，以及有关应用程序受监测流量的其他互联网流量指标图。

要开始探索应用程序流量的网络监测仪数据，请选择一个时间段。然后选择一个特定的地理位置（例如一个城市）以及（可选）其他筛选条件。网络监测仪会对您的数据应用筛选条件，然后您可以看到显示应用程序监测数据的图形。分析页面中包含的图形包括应用程序随时间变化的性能分数、可用性分数、传输的受监测字节数（适用于 VPC、网络负载均衡器和 CloudFront 分配）或客户端连接计数（适用于 WorkSpaces 目录）以及往返时间（RTT）。

分析页面顶部的选项决定了该页中图形所显示的时间范围和流量类型。您可以按客户端位置或 ASN 进行筛选，也可以选择按特定粒度显示流量图（默认为城市级别）。

主要客户端位置

默认情况下，主要客户端位置图会显示受监测靠前的位置。您可以选择其他字段对图形进行排序，也可以按其他方式对图形进行排序，例如按流量最低的位置进行排序。

您在该页面选择的筛选条件决定了这些位置的区域、时间范围等。

流量运行状况分数

互联网流量图

这一部分显示受监测流量的流量运行状况分数和指标图。这些图形反映了根据您在该页面顶部选择的筛选条件筛选后的数据。

运行状况分数图通过调用影响受监测客户端流量的运行状况事件来显示局部和整体流量的性能和可用性信息。对于不同 ASN 和 AWS 服务在不同地理位置之间的网络流量，AWS 拥有大量关于互联

网性能和可用性的历史数据。网络监测仪使用 AWS 从其全球网络足迹中捕获的连接数据来计算互联网流量的性能和可用性基准。这与我们在 AWS 用来监控我们自己的互联网正常运行时间和可用性的数据相同。

以这些测量值为基准，Internet Monitor 可以检测到应用程序的互联网性能和可用性与基准相比何时有所下降。为了更容易看到下降情况，我们以性能分数和可用性分数的形式向您报告此信息。有关更多信息，请参阅 AWS 如何计算性能和可用性分数。

其他的图形会显示应用程序流量的已传输受监测字节数（适用于 VPC、网络负载均衡器和 CloudFront 分配）或客户端连接计数（适用于 WorkSpaces 目录）以及往返时间（RTT）。

请注意，跨最终用户位置汇总往返时间（RTT）数据时，将根据每个客户端位置驱动的流量对该值进行加权。例如，假设有两个客户端位置，一个使用 5 毫秒 RTT 提供 90% 的流量，另一个使用 10 毫秒 RTT 提供 10% 的流量，则 RTT 汇总结果是 5.5 毫秒（计算过程：5 毫秒 * 0.9 + 10 毫秒 * 0.1）。

您还可以使用 CloudWatch 工具或其他方法探索网络监测仪存储的受监测流量的互联网监测数据。有关更多信息，请参阅 使用 CloudWatch 工具和网络监测仪查询接口探索您的数据。此外，您还可以创建基于网络监测仪数据的 CloudWatch 警报，（例如）用来通知您运行状况事件。有关更多信息，请参阅使用 Amazon CloudWatch 网络监测仪创建警报。

在网络监测仪中获取优化应用程序性能的建议（“优化”页面）

使用网络监测仪控制台中的优化页面，可以获取有关如何为客户端优化应用程序性能的建议。网络监测仪会评估您监测的应用程序流量，并确定是否可以通过更改您为应用程序配置的 AWS 区域来减少延迟。如果您选择在建议中包含 Amazon CloudFront，则还可以查看延迟变化。

您可以查看有关应用程序流量排名靠前的区域的建议，或者有关流量靠前的主要客户端位置的建议。

有关减少主要区域延迟的建议

为帮助您快速了解可减少客户端延迟的最佳选项，网络监测仪会自动为主要区域（按流量）提供改进应用程序延迟的建议。

您还可以探索有关应用程序服务客户端的所有区域的配置更改。这包括按更精细的粒度获取有关每项建议更改的详细信息，例如按特定的客户端位置。要探索应用程序的所有区域配置和预期延迟变化，请选择所有区域的优化建议。

有关减少所有区域延迟的建议

要探索有关可在客户端访问应用程序的所有区域减少延迟的建议，请选择所有区域的优化建议以打开新的控制面板页面。在此页面上，您可以选择要配置的不同区域，其中包含使用 CloudFront 进行配置比较的选项，然后比较每个选定配置的第一字节时间（TTFB）进行比较。

然后在每次比较时，您还可以查看一个包含更精细数据（按客户端位置）以及每个位置的平均预期 TTFB 的表格。

有关减少主要位置延迟的建议

网络监测仪还可按特定位置提供为客户端减少应用程序延迟的建议。当表格列出针对同一位置的多个建议时，展开该行的城市位置可以查看详细信息。

请注意，如果将配置更改为使用其他区域或使用 CloudFront，则延迟改善情况可能会因客户端位置而异。例如，某些位置的延迟可能会有所改善，但其他位置则不会变化甚至恶化。

通过查看此页面上的建议，您可以开始规划能够提高客户端性能的配置和部署。请注意，当数据不可显示时，您可能会在列中看到短划线（-）而不是值。

有关 TTFB 计算的更多信息，请参阅 TTFB 和延迟的 AWS 计算。要查看如何提高性能的具体示例，请参阅 Using Amazon CloudWatch Internet Monitor for a Better Gaming Experience。

在网络监测仪中监测详细信息（“配置”页面）

在配置页面上，您可以查看有关监测仪的详细信息，包括要监测流量的资源。与在网络监测仪控制面板的每个页面上一样，您可以选择编辑监测仪来更改监测仪的选项，包括添加或移除资源。有关本节中列出的选项，请参阅提供的链接，了解更改这些选项的具体步骤。

查看运行状况事件阈值

在这一部分中，您可以查看为此监测仪配置的运行状况事件的当前阈值。

要更新运行状况阈值，请参阅更改运行状况事件阈值。

查看和评估流量覆盖范围

在这一部分中，您可以比较更改应用程序的流量监测百分比对不同百分比所含城市-网络数量的影响。您还可以更改监测仪所监测的流量百分比，也可以更改监测仪所含的城市-网络数量限制。

有关详细步骤和信息，请参阅探索更改应用程序流量百分比的选项。

将互联网测量数据发布到 Amazon S3 的配置详细信息

如果您已将网络监测仪配置为将监测仪的互联网测量数据发布到 Amazon S3 存储桶，则此处会显示有关您的配置的信息。

要配置此选项，请参阅将互联网监测数据发布到 S3。

使用 CloudWatch 工具和网络监测仪查询接口探索您的数据

除了使用 Amazon CloudWatch 网络监测仪控制面板可视化应用程序的性能和可用性之外，您还可以使用多种方法来深入了解网络监测仪为您生成的数据。这些方法包括将 CloudWatch 工具与存储在 CloudWatch 日志文件中的网络监测仪数据一起使用，以及使用 Internet Monitor 查询接口。您可以使用的工具包括 CloudWatch Logs Insights、CloudWatch Metrics、CloudWatch Contributor Insights 和 Amazon Athena。您可以根据需要使用其中一些或全部工具以及控制面板来探索网络监测仪数据。

网络监测仪汇总了有关应用程序流量和每个 AWS 区域 流量的 CloudWatch 指标，包括总流量影响、可用性和往返时间等数据。这些数据将发布到 CloudWatch Logs，也可以与网络监测仪查询接口一起使用。有关地理粒度以及可供探索之信息的其他方面的详细信息各不相同。

Amazon CloudWatch 网络监测仪以 5 分钟为间隔为您的监测仪发布数据，然后以多种方式提供这些数据。下表列出了访问网络监测仪数据的场景，并描述了为每个场景收集的数据的特征。

功能	CloudWatch Logs	导出到 S3	查询接口	CloudWatch 控制面板
默认情况下启用	是	否	是	是
为其收集数据的城市网络数量	前 500 个（参见下面的注释）	全部	全部	全部
数据留存	受用户控制	受用户控制	30 天	30 天
为其收集数据的地理粒度	全部（城市-网络、都会区+网络、分区+网络、国家/地区+网络）	城市网络	全部（城市-网络、都会区+网络、分区+网络、国家/地区+网络）	全部（城市-网络、都会区+网络、分区+网络、国家/地区+网络）

功能	CloudWatch Logs	导出到 S3	查询接口	CloudWatch 控制面板
如何查询和筛选数据	将 CloudWatch Logs Insights 与 Amazon CloudWatch 网络监测仪结合使用	使用 Amazon Athena 查询 Amazon S3 日志中的互联网测量数据	使用 Amazon CloudWatch 网络监测仪查询接口	使用 Internet Monitor 控制面板进行监控和优化

注意：前 500 个测量值对应于城市网络；前 250 个对应于都会区+网络，前 100 个对应于分区+网络，前 50 个对应于国家/地区+网络。

本章介绍如何使用 CloudWatch 工具或网络监测仪查询接口来查询和探索您的数据，以及每种方法的示例。

内容

- 将 CloudWatch Logs Insights 与 Amazon CloudWatch 网络监测仪结合使用
- 将 CloudWatch Insights 与 Amazon CloudWatch 网络监测仪结合使用
- 将 CloudWatch Metrics 与 Amazon CloudWatch 网络监测仪结合使用
- 使用 Amazon Athena 查询 Amazon S3 日志中的互联网测量数据
- 使用 Amazon CloudWatch 网络监测仪查询接口

将 CloudWatch Logs Insights 与 Amazon CloudWatch 网络监测仪结合使用

Amazon CloudWatch 网络监测仪将可用性和往返时间的精细测量数据发布到 CloudWatch Logs，您可以使用 CloudWatch Logs Insights 查询来筛选特定城市或地理位置（客户端位置）、客户端 ASN（ISP）和 AWS 源位置的日志子集。

要详细了解网络监测仪中的客户端位置准确性，请参阅 网络监测仪中的地理位置信息和准确性。

本节中的示例可以帮助您创建 CloudWatch Logs Insights 查询，以了解有关您自己的应用程序流量测量数据和指标的更多信息。如果您在 CloudWatch Logs Insights 中使用这些示例，请将 *monitorName* 替换为您自己的监测仪名称。

查看流量优化建议

在 Internet Monitor 的流量洞察选项卡上，您可以按位置筛选并查看流量优化建议。要查看此选项卡上的流量优化建议部分中显示的相同信息，但不使用位置粒度筛选条件，您可以使用以下 CloudWatch Logs Insights 查询。

1. 在 AWS Management Console 中，导航到 CloudWatch Logs Insights。
2. 对于 Log Group（日志组），选择 /aws/internet-monitor/*monitorName*/byCity 和 /aws/internet-monitor/*monitorName*/byCountry，然后指定时间范围。
3. 添加以下查询，然后运行该查询。

```
fields @timestamp,
clientLocation.city as @city, clientLocation.subdivision as @subdivision,
 clientLocation.country as @country,
`trafficInsights.timeToFirstByte.currentExperience.serviceName` as @serviceNameField,
concat(@serviceNameField, ` (`, `serviceLocation`, `)`) as @currentExperienceField,
concat(`trafficInsights.timeToFirstByte.ec2.serviceName`, ` (`,
 `trafficInsights.timeToFirstByte.ec2.serviceLocation`, `)`) as @ec2Field,
`trafficInsights.timeToFirstByte.cloudfront.serviceName` as @cloudfrontField,
concat(`clientLocation.networkName`, ` (AS`, `clientLocation.asn`, `)`) as @networkName
| filter ispresent(`trafficInsights.timeToFirstByte.currentExperience.value`)
| stats avg(`trafficInsights.timeToFirstByte.currentExperience.value`) as @averageTTFB,
avg(`trafficInsights.timeToFirstByte.ec2.value`) as @ec2TTFB,
avg(`trafficInsights.timeToFirstByte.cloudfront.value`) as @cloudfrontTTFB,
sum(`bytesIn` + `bytesOut`) as @totalBytes,
latest(@ec2Field) as @ec2,
latest(@currentExperienceField) as @currentExperience,
latest(@cloudfrontField) as @cloudfront,
count(*) by @networkName, @city, @subdivision, @country
| display @city, @subdivision, @country, @networkName, @totalBytes, @currentExperience,
 @averageTTFB, @ec2, @ec2TTFB, @cloudfront, @cloudfrontTTFB
| sort @totalBytes desc
```

查看互联网可用性和 RTT（p50、p90 和 p95）

要查看流量的互联网可用性和往返时间（p50、p90 和 p95），您可以使用以下 CloudWatch Logs Insights 查询。

最终用户地理位置：美国伊利诺伊州芝加哥

最终用户网络（ASN）：AS7018

AWS 服务位置：美国东部（弗吉尼亚州北部）地区

要查看日志，请执行以下操作：

1. 在 AWS Management Console 中，导航到 CloudWatch Logs Insights。
2. 对于 Log Group（日志组），选择 /aws/internet-monitor/*monitorName*/byCity 和 /aws/internet-monitor/*monitorName*/byCountry，然后指定时间范围。
3. 添加以下查询，然后运行该查询。

该查询将返回在选定时间范围内从伊利诺伊州芝加哥的 AS7018 连接到美国东部（弗吉尼亚州北部）地区用户的所有性能数据。

```
fields @timestamp,
internetHealth.availability.experienceScore as availabilityExperienceScore,
internetHealth.availability.percentageOfTotalTrafficImpacted as
 percentageOfTotalTrafficImpacted,
internetHealth.performance.experienceScore as performanceExperienceScore,
internetHealth.performance.roundTripTime.p50 as roundTripTimep50,
internetHealth.performance.roundTripTime.p90 as roundTripTimep90,
internetHealth.performance.roundTripTime.p95 as roundTripTimep95
 | filter clientLocation.country == `United States`
 and clientLocation.city == `Chicago`
 and serviceLocation == `us-east-1`
 and clientLocation.asn == 7018
```

有关更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。

将 CloudWatch Insights 与 Amazon CloudWatch 网络监测仪结合使用

CloudWatch Contributor Insights 可以帮助您确定应用程序的主要客户端位置和网络（ASN 或互联网服务提供商）。通过以下 Contributor Insights 示例规则，开始使用对 Amazon CloudWatch 网络监测仪有用的规则。有关更多信息，请参阅 创建 Contributor Insights 规则。

要详细了解网络监测仪中的客户端位置准确性，请参阅 网络监测仪中的地理位置信息和准确性。

> **Note**
>
> 网络监测仪每五分钟存储一次互联网监测数据，因此在设置 Contributor Insights 规则后，必须将查看图表的周期调整为五分钟。

查看受可用性影响的排名靠前的位置和 ASN

要查看受可用性下降影响的主要客户端位置和 ASN，您可以在语法编辑器中使用以下 Contributor Insights 规则。请将 *monitor-name* 替换为您自己的监视器名称。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.availability.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

查看受延迟影响的主要客户端位置和 ASN

要查看受往返时间（延迟）增加影响的主要客户端位置和 ASN，您可以在语法编辑器中使用以下 Contributor Insights 规则。请将 *monitor-name* 替换为您自己的监视器名称。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
```

```
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.performance.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

查看受总流量百分比影响的主要客户端位置和 ASN

要查看受总流量百分比影响的主要客户端位置和 ASN，您可以在语法编辑器中使用以下 Contributor Insights 规则。请将 *monitor-name* 替换为您自己的监视器名称。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.percentageOfTotalTraffic"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

将 CloudWatch Metrics 与 Amazon CloudWatch 网络监测仪结合使用

Amazon CloudWatch 网络监测仪会将指标发布到您的账户，包括性能、可用性、往返时间和吞吐量（每秒字节数）等指标，您可以在 CloudWatch 控制台的 CloudWatch Metrics 中查看。要查找监测仪的所有指标，请在 CloudWatch Metrics 控制面板中使用自定义命名空间 `AWS/InternetMonitor`。

这些指标跨越以下流量进行了汇总：监测仪中的 VPC、网络负载均衡器、CloudFront 分配或 WorkSpaces 目录的所有互联网流量，以及流向每个 AWS 区域 和受监测的互联网边缘站点的所有流量。区域由服务位置定义，服务位置可以是所有位置，也可以是一个特定的区域，例如 `us-east-1`。

注意：城市-网络是客户端位置和 ASN [通常是互联网服务提供商（ISP）]。

Internet Monitor 提供以下指标。

指标	描述
PerformanceScore	性能分数表示未出现性能下降的流量的估计百分比。
AvailabilityScore	可用性分数表示未出现可用性下降的流量的估计百分比。
BytesIn	在所有应用程序城市网络上为您的应用程序互联网流量传入的字节。
BytesOut	在所有应用程序城市网络上为您的应用程序互联网流量传出的字节。
BytesInMonitored	在受监测的城市网络上为您的应用程序互联网流量传入的字节。
BytesOutMonitored	在受监测的城市网络上为您的应用程序互联网流量传出的字节。
往返时间（RTT）	AWS 区域、ASN（通常是互联网服务提供商或 ISP）和特定于 VPC、网络负载均衡器、CloudFront 分配或 WorkSpaces 目录的位置（例如城市）之间的往返时间。

指标	描述
CityNetworksMonitored	Internet Monitor 监测的应用程序互联网流量的城市网络数量。此值永远不会超过您为监测仪设置的城市网络最大上限。
TrafficMonitoredPercent	此监测仪的应用程序互联网总流量的百分比，由 Internet Monitor 正在监测的城市网络表示（包含在内）。如果客户端访问应用程序的城市网络数量超过了您为监测仪设置的城市网络最大限制，则该值将小于 100（即小于 100%）。
CityNetworksFor100PercentTraffic	如果您想在 Internet Monitor 中监测 100% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor99PercentTraffic	如果您想在 Internet Monitor 中监测 99% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor95PercentTraffic	如果您想在 Internet Monitor 中监测 95% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor90PercentTraffic	如果您想在 Internet Monitor 中监测 90% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor75PercentTraffic	如果您想在 Internet Monitor 中监测 75% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor50PercentTraffic	如果您想在 Internet Monitor 中监测 50% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。
CityNetworksFor25PercentTraffic	如果您想在 Internet Monitor 中监测 25% 的应用程序互联网流量，则应将城市网络最大限制设置为此数字。

> **Note**
>
> 要查看使用其中几个指标来帮助确定要为监测仪选择的城市网络最大值的示例，请参阅选择城市网络最大值。

有关更多信息，请参阅 Amazon CloudWatch 指标。

使用 Amazon Athena 查询 Amazon S3 日志中的互联网测量数据

您可以使用 Amazon Athena，来查询和查看 Amazon CloudWatch 网络监测仪发布到 Amazon S3 存储桶的互联网测量数据。网络监测仪中有一个选项，选择此选项后可将应用程序的互联网测量数据发布到 S3 存储桶，以了解所监测城市-网络 [客户端位置和 ASN，后者通常是互联网服务提供商（ISP）] 的互联网流量。无论您是否选择将测量数据发布到 S3，Internet Monitor 每五分钟都会自动将每个监测仪的前 500 个（按流量计）城市网络的互联网测量数据发布到 CloudWatch Logs。

本章包括如何在 Athena 中为位于 S3 日志文件中的互联网测量数据创建表的步骤，然后提供示例查询以查看测量数据的不同视图。例如，您可以按延迟影响查询受影响的前 10 个城市网络。

使用 Amazon Athena 在 Internet Monitor 中创建互联网测量数据表

要开始将 Athena 与您的网络监测仪 S3 日志文件结合使用，您首先需要为互联网测量数据创建一个表。

按照此过程中的步骤在 Athena 中根据 S3 日志文件创建表。然后，您可以在表上运行 Athena 查询，例如这些示例互联网测量数据查询，以获取有关测量数据的信息。

创建 Athena 表

1. 从 https://console.aws.amazon.com/athena/ 打开 Athena 控制台。
2. 在 Athena 查询编辑器中，输入查询语句以生成一个包含 Internet Monitor 互联网测量数据的表。将 LOCATION 参数的值替换为存储 Internet Monitor 互联网测量数据的 S3 存储桶的位置。

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
```

```
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

3. 输入一条语句来创建一个分区以读取数据。例如，以下示例查询创建了指定日期和位置的单个分区：

```
ALTER TABLE internet_measurements
ADD PARTITION (year = 'YYYY', month = 'MM', day = 'dd')
LOCATION
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/YYYY/
MM/DD';
```

4. 选择运行。

互联网测量数据的 Athena 语句示例

下面是用于生成表的语句示例：

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

以下是用于创建分区以读取数据的语句示例：

```
ALTER TABLE internet_measurements
ADD PARTITION (year = '2023', month = '04', day = '07')
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/2023/04/07/'
```

用于 Internet Monitor 中互联网测量数据的 Amazon Athena 查询示例

本节包含查询示例，您可以在 Amazon Athena 中使用这些查询来获取有关发布到 Amazon S3 的应用程序互联网测量数据的信息。

查询前 10 个（按流量的总百分比计）受影响的客户端位置和 ASN

运行此 Athena 查询，返回受影响的前 10 个（按流量的总百分比计）城市网络（即客户端位置和 ASN，通常是互联网服务提供商）。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(percentageoftotaltraffic) as percentageoftotaltraffic
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageoftotaltraffic desc
limit 10
```

查询受影响的前 10 个（按可用性计）客户端位置和 ASN

运行此 Athena 查询，返回受影响的前 10 个（按流量的总百分比计）城市网络（即客户端位置和 ASN，通常是互联网服务提供商）。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.availability.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
```

```
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

查询受影响的前 10 个（按延迟计）客户端位置和 ASN

运行此 Athena 查询，返回受影响的前 10 个（按延迟影响计）城市网络（即客户端位置和 ASN，通常是互联网服务提供商）。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.performance.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

查询您的客户端位置和 ASN 的流量亮点

运行此 Athena 查询以返回流量亮点，包括可用性分数、性能分数以及城市网络（即客户端位置和 ASN，通常是互联网服务提供商）的首字节时间。

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.subdivision') as subdivision,
    json_extract_scalar(clientLocation, '$.country') as country,
    avg(cast(json_extract_scalar(internetHealth, '$.availability.experiencescore') as
 double)) as availabilityScore,
    avg(cast(json_extract_scalar(internetHealth, '$.performance.experiencescore') as
 double)) performanceScore,
    avg(cast(json_extract_scalar(trafficinsights,
 '$.timetofirstbyte.currentexperience.value') as double)) as averageTTFB,
    sum(bytesIn) as bytesIn,
```

```
    sum(bytesOut) as bytesOut,
    sum(bytesIn + bytesOut) as totalBytes
FROM internet_measurements
where json_extract_scalar(clientLocation, '$.city') != 'N/A'
GROUP BY
json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.subdivision'),
    json_extract_scalar(clientLocation, '$.country')
ORDER BY totalBytes desc
limit 100
```

有关使用 Athena 的更多信息，请参阅 Amazon Athena 用户指南。

使用 Amazon CloudWatch 网络监测仪查询接口

要详细了解 AWS 应用程序的互联网流量，可以选择使用 Amazon CloudWatch 网络监测仪查询接口。要使用该查询接口，您需要使用所选数据筛选条件创建查询，然后运行查询以返回网络监测仪数据的子集。浏览查询返回的数据可以让您深入了解应用程序在互联网上的表现。

您可以查询和探索网络监测仪使用监测仪捕获的所有指标，包括可用性和性能分数、传输的字节数、往返时间和首字节时间（TTFB）。

网络监测仪使用查询接口提供您可以在网络监测仪控制台控制面板中探索的数据。通过使用控制面板中的搜索选项（在分析页面或优化页面上），您可以查询和筛选应用程序的互联网数据。

如果您想要比控制面板更灵活的数据探索和筛选方式，则可以通过将网络监测仪 API 操作与 AWS Command Line Interface 或 AWS SDK 配合使用，自行使用查询接口。本部分介绍可与查询接口一起使用的查询类型，以及为创建数据子集可指定的筛选条件器，以便深入了解应用程序的互联网流量。

主题

- 如何使用查询接口
- 查询示例
- 获取查询结果
- 故障排除

如何使用查询接口

您可以使用查询接口创建查询，方法是选择查询类型，然后指定筛选条件值，以返回所需的特定日志文件数据子集。然后，您可以使用该数据子集执行进一步筛选和排序、创建报告等操作。

查询过程的工作方式如下所示：

1. 运行查询时，网络监测仪会返回查询所特有的 `query ID`。本部分介绍可用的查询类型以及可用于筛选查询中数据的选项。要了解其工作原理，您还可以查看查询示例部分。
2. 您可以搭配使用监测仪名称和 GetQueryResults API 操作来指定查询 ID，以返回查询的数据结果。每种查询类型都会返回不同的数据字段集。要了解更多信息，请参阅获取查询结果。

查询接口提供以下 3 种查询类型。每种查询类型都会从日志文件中返回一组不同的流量信息，如下所示。

- 测量数据：以 5 分钟为间隔提供可用性分数、性能分数、总流量和往返时间。
- 排名靠前的位置：按流量提供您正在监测的“排名靠前的位置 + ASN”组合的可用性分数、性能分数、总流量和首字节时间（TTFB）信息。
- 排名靠前的位置详情：以 1 小时为间隔提供 Amazon CloudFront 的 TTFB、您当前的配置以及性能最佳的 Amazon EC2 配置。
- 总体流量建议：提供每个受监测 AWS 位置的所有流量的 TTFB（基于 30 天加权平均值）。
- 总体流量建议详细信息：提供了每个主要位置以及建议的 AWS 位置的 TTFB（基于 30 天加权平均值）。

对于上述每种查询类型，您可以通过指定以下一个或多个条件，对数据进行更多筛选：

- AWS 位置：对于 AWS 位置，您可以指定 CloudFront 或 AWS 区域，例如 `us-east-2`、`us-west-2` 等。
- ASN：指定 ASN，它通常是互联网服务提供商（ISP）。
- 客户端位置：对于位置，请指定城市、都会区、分区或国家/地区。
- 地理位置：为某些查询指定 `geo`。对于使用 `Top locations` 查询类型的查询，此为必需项，但不允许用于其他查询类型。要了解何时为筛选参数指定 `geo`，请参阅 Query examples 部分。

可用于筛选数据的运算符是 `EQUALS` 和 `NOT_EQUALS`。有关筛选参数的详细信息，请参阅 FilterParameter API 操作。

要查看有关查询接口操作的详细信息，请参阅《Amazon CloudWatch 网络监测仪 API 参考指南》中的以下 API 操作：

- 要创建和运行查询，请参阅 StartQuery API 操作。

- 要停止查询，请参阅 StopQuery API 操作。
- 要为您创建的查询返回数据，请参阅 GetQueryResults API 操作。
- 要检索查询的状态，请参阅 GetQueryStatus API 操作。

查询示例

要创建可用于从监测仪日志文件中检索经过筛选的数据集的查询，可以使用 StartQuery API 操作。您可以为查询指定查询类型和筛选参数。然后，当您使用网络监测仪查询接口 API 操作通过查询获取查询结果时，它将检索您要使用的数据子集。

为了说明查询类型和筛选参数的工作原理，让我们看一些示例。

示例 1

假设您想要检索特定国家/地区除一个城市外的所有监测仪日志文件数据。以下示例显示了一个查询的筛选参数，您可以使用 `StartQuery` 操作为此场景创建该查询。

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "MEASUREMENTS"
   FilterParameters: [
      {
       Field: "country",
       Operator: "EQUALS",
       Values: ["Germany"]
      },
      {
       Field: "city",
       Operator: "NOT_EQUALS",
       Values: ["Berlin"]
      },
    ]
}
```

示例 2

再举个示例，假设您想按大都会地区查看排名靠前的位置。您可以为此场景使用以下示例查询。

```
{
```

```
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["metro"]
      },
    ]
}
```

示例 3

现在，假设您想查看洛杉矶都会区的主要城市网络组合。为此，请指定 geo=city，然后将 metro 设置为“洛杉矶”。现在，该查询返回的是洛杉矶都会区排名靠前的城市网络，而不是总体上排名靠前的都会区+网络。

以下是您可以使用的示例查询：

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["city"]
      },
      {
       Field: "metro",
       Operator: "EQUALS",
       Values: ["Los Angeles"]
      }
    ]
}
```

示例 4。

然后，假设您要检索特定地区（例如美国某州）的 TTFB 数据。

您可以为此场景使用以下示例查询：

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATION_DETAILS"
   FilterParameters: [
      {
       Field: "subdivision",
       Operator: "EQUALS",
       Values: ["California"]
      },
    ]
}
```

示例 5

现在，假设您要检索应用程序具有客户端流量的每个位置的 TTFB 数据。

您可以为此场景使用以下示例查询：

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS"
   FilterParameters: []
}

Results:
[us-east-1, 40, us-west-2, 30],
[us-east-1, 40, us-west-1, 35],
[us-east-1, 40, us-east-1, 44],
[us-east-1, 40, CloudFront, 22],
...
[us-east-2, 44, us-west-2, 30],
[us-east-2, 44, us-west-1, 35],
...
```

示例 6

最后，假设您要检索特定新 AWS 区域的 TTFB 数据。

您可以为此场景使用以下示例查询：

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS_DETAILS"
   FilterParameters: [
      {
       Field: "proposed_aws_location",
       Operator: "EQUALS",
       Values: ["us-west-2"]
      },
   ]
}

Results:
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-east-1, 40, 350,
 350, us-west-2, 45]
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-west-1, 35, 450,
 450, us-west-2, 45]
```

获取查询结果

定义查询后，您可以通过运行另一个网络监测仪 API 操作 GetQueryResults 来返回一组查询结果。运行 GetQueryResults 时 ，您可以为已定义的查询指定查询 ID 以及监测仪名称。GetQueryResults 会将指定查询的数据检索到结果集中。

运行查询时，请确保查询已完成运行，然后再使用 GetQueryResults 查看结果。您可以使用 GetQueryStatus API 操作来确定查询是否已完成。当查询的 Status 为 SUCCEEDED 时，您可以放心查看结果。

查询完成后，您可以使用以下信息帮助查看结果。您用于创建查询的每种查询类型都包含一组来自日志文件的唯一数据字段，如以下列表中所述：

测量值

measurements 查询类型会返回以下数据：

```
timestamp, availability, performance, bytes_in, bytes_out, rtt_p50,
rtt_p90, rtt_p95
```

排名靠前的位置

top locations 查询类型会按位置对数据进行分组，并提供一段时间内的平均数据。它返回的数据包括以下各项：

aws_location, city, metro, subdivision, country, asn, availability, performance, bytes_in, bytes_out, current_fbl, best_ec2, best_ec2_region, best_cf_fbl

请注意，仅当您为 geo 字段选择该位置类型时，才会返回 city、metro 和 subdivision。根据您为 geo 指定的位置类型，将返回以下位置字段：

```
city = city, metro, subdivision, country
metro = metro, subdivision, country
subdivision = subdivision, country
country = country
```

排名靠前的位置详细信息

top locations details 查询类型会返回按小时分组的数据。该查询会返回以下数据：

timestamp, current_service, current_fbl, best_ec2_fbl, best_ec2_region, best_cf_fbl

运行 GetQueryResults API 操作时，网络监测仪会在响应中返回以下各项：

- 包含查询返回的结果的数据字符串数组。该信息以与 Fields 字段匹配的数组形式返回，也由该 API 调用返回。您可以使用 Fields 字段解析 Data 存储库中的信息，然后根据需要对其进行进一步筛选或排序。
- 一个字段数组，其中列出了查询为其返回数据的字段（在 Data 字段响应中）。数组中的每一项都是一个名称-数据类型对，例如 availability_score-float。

故障排除

如果您在使用查询接口 API 操作时返回错误，请确认您拥有使用 Amazon CloudWatch 网络监测仪所需的权限。具体而言，请确保您拥有以下权限：

```
internetmonitor:StartQuery
internetmonitor:GetQueryStatus
internetmonitor:GetQueryResults
```

```
internetmonitor:StopQuery
```

这些权限包含在推荐 AWS Identity and Access Management 策略中，以便在控制台中使用网络监测仪控制面板。有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的 IAM 权限。

使用 Amazon CloudWatch 网络监测仪创建警报

您可以根据 Amazon CloudWatch 网络监测仪指标创建 Amazon CloudWatch 告警，就为其他 Amazon CloudWatch Internet Monitor 指标创建告警一样。

例如，您可以根据网络监测仪指标 `PerformanceScore` 创建警报，并将其配置为在指标低于选定值时发送通知。您可以遵循与其他 CloudWatch 指标相同的指南为 Internet Monitor 指标配置警报。

以下示例演示了您可以选择为其创建告警的网络监测仪指标：

- PerformanceScore
- AvailabilityScore
- RoundtripTime

要查看网络监测仪的所有可用指标，请参阅 将 CloudWatch Metrics 与 Amazon CloudWatch 网络监测仪结合使用。

以下过程提供了一个示例，通过导航到 CloudWatch 控制面板中的指标以在 PerformanceScore 上设置告警。然后，您按照标准 CloudWatch 步骤根据您选择的阈值创建告警，并设置通知或选择其他选项。

在 CloudWatch 指标中为 PerformanceScore 创建告警

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 选择指标，然后选择所有指标。
3. 通过选择 `AWS/InternetMonitor` 筛选 Internet Monitor。
4. 选择 MeasurementSource、MonitorName。
5. 在列表中，选择 PerformanceScore。
6. 在 GraphedMetrics 选项卡的操作下，选择钟形图标以基于静态阈值创建警报。

现在，按照标准的 CloudWatch 步骤为告警选择选项。例如，如果 PerformanceScore 低于特定阈值，您可以选择通过 Amazon SNS 消息接收通知。或者，或者除此之外，您可以将告警添加到控制面板。

记住以下内容：

- 网络监测仪指标通常在 20 分钟内计算并发布。
- 在根据网络监测仪指标创建告警时，请务必在设置警报的回顾期时，考虑发布前的短暂延迟。我们建议您将评估期配置为有至少 25 分钟的回顾期。

要详细了解如何将 CloudWatch 警报与网络监测仪结合使用，请参阅以下博文：Using Amazon CloudWatch Internet Monitor for enhanced internet observability。

有关创建 CloudWatch 告警时的选项的更多信息，请参阅 根据静态阈值创建 CloudWatch 告警。

将 Amazon CloudWatch 网络监测仪与 Amazon EventBridge 结合使用

Amazon CloudWatch 网络监测仪针对网络问题创建的运行状况事件通过 Amazon EventBridge 发布，因此您可以发送有关应用程序最终用户体验下降的通知。

要使用 EventBridge 处理 Internet Monitor 运行状况事件，请遵循此处的指南。

在 EventBridge 中为 Internet Monitor 设置规则

1. 在 AWS Management Console 的 EventBridge 中，选择 Rules（规则），然后输入名称和描述。在 Default（默认）事件总线上创建规则。
2. 在步骤 2 中，选择其他作为事件源，然后在事件模式下匹配以下源。

```
{
  "source": ["aws.internetmonitor"]
}
```

3. 在步骤 3 中，对于目标，选择 AWS 服务和 CloudWatch Logs 组，然后选择一个现有的日志组或创建一个新的日志组。
4. 添加任何所需的标签，然后创建规则。这将使用来自 EventBridge 的事件填充您选择的 CloudWatch 日志组。

有关 EventBridge 规则如何与事件模式配合使用的更多信息，请参阅《Amazon EventBridge 用户指南》中的 Amazon EventBridge 事件模式。

排查 CloudWatch 日志和指标访问错误

要支持某些功能，Amazon CloudWatch 网络监测仪必须与包括日志和指标在内的特定 Amazon CloudWatch 资源进行交互。如果网络监测仪无法访问其需要访问的 CloudWatch 资源，则监测仪的状态代码将设置为 FAULT_ACCESS_CLOUDWATCH。

多个原因可造成监测仪处于状态 FAULT_ACCESS_CLOUDWATCH。以下各部分列出了这些错误的可能原因以及建议的故障排除步骤。

网络监测仪无法访问您账户中的 CloudWatch 日志

网络监测仪会发布有关受监测应用程序流量的诊断日志。它将这些日志发布到位于 `/aws/internet-monitor/monitor_name/[byCity|byMetro|bySubdivision|byCountry]` 位置的 CloudWatch 日志中的日志组。互联网监测仪无法访问这些日志组。

错误状态和可能的解决方案：

- PutLogEvents 节流错误：互联网监测仪服务在尝试将监测仪日志发布到 CloudWatch 时可能已受到节流。查看您账户的节流限制，如有必要，可以申请提高限额。
- 未找到日志组：禁用监测仪，然后重新启用。启用监测仪可重新启动日志组创建，这或许可以解决问题。
- PutLogEvents 访问遭拒错误：联系 AWS 支持人员以获取帮助。
- PutLogEvents 未知或一般错误：联系 AWS 支持人员以获取帮助。

网络监测仪无法访问您账户中的 CloudWatch 指标

网络监测仪就监测仪跟踪的应用程序流量提供特定 CloudWatch 指标。网络监测仪尝试向 CloudWatch 提供这些指标时出现错误。

错误状态和可能的解决方案：

- PutMetricData 节流错误：互联网监测仪服务在尝试将监测仪指标发布到 CloudWatch 时可能已受到节流。查看您账户的节流限制，如有必要，可以申请提高限额。
- PutMetricData 访问遭拒错误：联系 AWS 支持人员以获取帮助。
- PutMetricData 未知或一般错误：联系 AWS 支持人员以获取帮助。

Amazon CloudWatch 网络监测仪的数据保护和数据隐私

AWS 责任共担模式适用于 Amazon CloudWatch 网络监测仪中的数据保护和数据隐私。如该模式中所述，AWS 负责保护运行所有 AWS 云的全球基础设施。您负责维护对托管在此基础架构上的内容的控制。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全博客上的 AWS 责任共担模式和 GDPR 博客文章。有关遵守 GDPR 要求的更多资源，请参阅一般数据保护条例 (GDPR) 中心。

强烈建议您不要在自由格式字段中输入敏感标识信息（例如终端用户客户账户号码、邮箱地址或其他个人信息）。您输入到 Amazon CloudWatch 网络监测仪或其他服务中的任何数据都可能包含在诊断日志中。

适用于 Amazon CloudWatch 网络监测仪的 Identity and Access Management

AWS Identity and Access Management（IAM）是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和获得授权（具有权限）来使用 Internet Monitor 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

Important

2024 年 7 月 8 日 Internet Monitor 的资源更改
如果您在 2024 年 7 月 8 日之前创建了包含 Internet Monitor 资源的 IAM 策略，请注意 Internet Monitor 资源和资源类型的以下更改：

- 仅在 Monitor 资源类型上支持 GetHealthEvent 操作的资源级权限。HealthEvent 资源不支持这些权限。

要查看有关您可以在策略中指定的操作、资源和条件键以管理对 Internet Monitor 中 AWS 资源访问的更多信息，请参阅 Amazon CloudWatch 网络监测仪的操作、资源和条件键。

内容

- 将 IAM 策略升级到 IPv6
- Amazon CloudWatch 网络监测仪如何与 IAM 协同工作
- Amazon CloudWatch 网络监测仪的 AWS 托管策略
- Amazon CloudWatch 网络监测仪的 IAM 权限
- Amazon CloudWatch 网络监测仪的服务相关角色

将 IAM 策略升级到 IPv6

Amazon CloudWatch 网络监测仪客户使用 IAM 策略来设置允许的 IP 地址范围，以阻止配置范围之外的任何 IP 地址访问网络监测仪 API。

internetmonitor.*region*.api.aws 端点正在升级为支持双栈配置（IPv4 和 IPv6）。

未更新为支持处理 IPv6 地址的 IP 地址筛选策略可能导致客户端无法访问网络监测仪 API。

受 IPv6 升级影响的客户

使用双栈配置且策略中包含 aws:sourceIp 筛选条件的客户会受到本次升级的影响。双栈配置意味着网络同时支持 IPv4 和 IPv6。

如果您使用双栈配置，则必须更新当前使用 IPv4 格式地址配置的 IAM 策略，使其包含 IPv6 格式的地址。

以下总结了建议的操作，具体取决于您的情况。要确认 SDK 使用的端点，请参阅识别代码使用的网络监测仪端点。

终端节点	使用带 **aws:sourceIp** 条件的 IAM 策略？	推荐操作
`internetmonitor.region.amazonaws.com`（非双栈）	是	要限制为仅允许 IPv4 访问，则无需执行进一步的操作。如果预计未来将需要 IPv6 支持，则可以采取措施确保同时兼容 IPv4 和 IPv6。 为确保未来的兼容性，请在 2024 年 11 月 1 日当天或之后更新 SDK，然后通过设置 `useDualstackEndpoint=true` 来更新应用程序以使用双栈端点。有关更多信息，请参阅 Dual-stack and FIPS endpoints。 如果您选择同时使用 IPv4 和 IPv6，则还必须更新 IAM 策略中的 IP 地址筛选条件 (`aws:sourceIp`)，使其包括 IPv6 地址。
	否	

终端节点	使用带 **aws:sourceIp** 条件的 IAM 策略？	推荐操作
`internetmonitor.region.amazonaws.com` （非双栈）		要限制为仅允许 IPv4 访问，则无需执行进一步的操作。如果预计未来将需要 IPv6 支持，则可以采取措施确保同时兼容 IPv4 和 IPv6。 为确保未来的兼容性，请在 2024 年 11 月 1 日当天或之后更新 SDK，然后通过设置 `useDualstackEndpoint=true` 来更新应用程序以使用双栈端点。有关更多信息，请参阅 Dual-stack and FIPS endpoints。

终端节点	使用带 **aws:sourceIp** 条件的 IAM 策略？	推荐操作
`internetmonitor.region.api.aws`	是	目前，此端点仅支持 IPV4。从 2024 年 11 月 1 日起，此端点上将会启用 IPv6。 为确保未来同时兼容 IPv4 和 IPv6，请在 2024 年 11 月 1 日当天或之后更新 SDK，然后通过设置 `useDualstackEndpoint=true` 来更新应用程序以使用双栈端点。有关更多信息，请参阅 Dual-stack and FIPS endpoints。 更改为同时使用 IPv4 和 IPv6 时，还必须更新 IAM 策略中的 IP 地址筛选条件 (`aws:sourceIp`)，使其包括 IPv6 地址。 如果想限制为仅允许 IPv4 访问，请设置 `useDualstackEndpoint=false` 。有关更多信息，请参阅 Dual-stack and FIPS endpoints。

终端节点	使用带 **aws:sourceIp** 条件的 IAM 策略？	推荐操作
`internetmonitor.region.api.aws`	否	目前，此端点仅支持 IPV4。从 2024 年 11 月 1 日起，此端点上将会启用 IPv6。 为确保未来同时兼容 IPv4 和 IPv6，请在 2024 年 11 月 1 日当天或之后更新 SDK，然后通过设置 `useDualstackEndpoint=true` 来更新应用程序以使用双栈端点。有关更多信息，请参阅 Dual-stack and FIPS endpoints。 如果想限制为仅允许 IPv4 访问，请设置 `useDualstackEndpoint=false` 。有关更多信息，请参阅 Dual-stack and FIPS endpoints 。

有关访问问题的帮助，请联系 AWS Support。

什么是 IPv6？

IPv6 是下一代 IP 标准，旨在最终取代 IPv4。IPv4 使用 32 位寻址方案，最多支持 43 亿个设备。IPv6 使用 128 位寻址方案，最多可支持大约 340 万亿（即 2 的 128 次方）个设备。

以下是 IPv6 地址的示例：

```
2001:cdba:0000:0000:0000:0000:3257:9652
2001:cdba:0:0:0:0:3257:9652
2001:cdba::3257:965
```

IPv6 提供了更大的地址空间、更高的路由效率以及对新互联网服务的更好支持。通过更新到双栈配置并支持 IPv6，网络监测仪可以提高性能和可扩展性。按照本节中的步骤更新配置并利用双栈支持的优势。

识别代码使用的网络监测器端点

如果您使用网络监测仪 SDK，请首先验证代码正在使用的端点：IPv4 端点还是双栈（IPv4 和 IPv6）端点。如果未将 SDK 与网络监测仪结合使用，则可跳过本节。

您可以运行以下代码示例来确定正在使用的网络监测仪端点。在此例中，我们将在美国东部（弗吉尼亚州北部）区域中使用适用于 Go 的网络监测仪 SDK。

```
package main

import (
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/internetmonitor"
)

func main() {
    // Create a new session with the default configuration
    sess := session.Must(session.NewSession(&aws.Config{
        Region: aws.String("us-east-1"),
    }))

    // Create a new Internet Monitor client
    internetMonitorClient := internetmonitor.New(sess)

    // Get the endpoint URL
    endpoint := internetMonitorClient.Endpoint

    fmt.Printf("Internet Monitor endpoint URL: %s\n", endpoint)
}
```

当您运行此代码时，将会返回网络监测仪端点。如果您看到以下响应，则表示您使用的网络监测仪域仅支持 IPv4。您可以确定这一点，是因为端点 URL 格式中包含 amazonaws.com。

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.amazonaws.com
```

如果您看到以下响应，则表示您使用的域正在升级为支持双栈（IPv4 和 IPv6）。此处您可以确定这一点，是因为端点 URL 中包含 api.aws。但请注意，在升级完成之前，此端点仅支持 IPv4。

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.api.aws
```

更新 IAM 策略以支持 IPv6

IAM 策略使用 aws:SourceIp 筛选条件来设置允许的 IP 地址范围。

双栈配置同时支持 IPv4 和 IPv6 流量。如果您的网络使用双栈配置，则必须确保用于 IP 地址筛选的所有 IAM 策略已更新为包括 IPv6 地址范围。

例如，此策略允许 Condition 元素中列出的 IPv4 地址范围 192.0.2.0.* 和 203.0.113.0.*。

```
# https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_examples_aws_deny-ip.html
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*",
                    "*203.0.113.0/24*"
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
    }
}
```

为更新此策略，我们将更改策略的 Condition 元素以添加 IPv6 地址范围，如以下示例所示：

```
"Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*", <<Existing IPv4 address - DO NOT REMOVE>>
                    "*203.0.113.0/24*", <<Existing IPv4 address  - DO NOT REMOVE>>
                    "*2001:DB8:1234:5678::/64*", <<New IPv6 IP address>>
```

```
                "*2001:cdba:3257:8593::/64*" <<New IPv6 IP address>>
            ]
        },
        "Bool": {
            "aws:ViaAWSService": "false"
        }
    }
```

> **Important**
>
> 请勿删除策略中现有的 IPv4 地址。向后兼容性将需要这些地址。

有关使用 IAM 管理访问权限的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的托管策略与内联策略。

更新策略后测试网络

更新 IAM 策略以包括对 IPv6 地址的支持后，我们建议您测试网络是否能够访问 IPv6 端点。本节提供了多个示例，具体取决于您使用的操作系统。

使用 Linux/Unix 或 Mac OS X 测试网络

如果使用的是 Linux/Unix 或 Mac OS X，可以使用以下 curl 命令来测试您的网络是否能够访问 IPv6 端点。

```
curl -v -s -o /dev/null http://ipv6.ec2-reachability.amazonaws.com/
```

如果通过 IPv6 建立了连接，则连接的 IP 地址会显示与以下类似的信息：

```
* About to connect() to aws.amazon.com port 443 (#0)
*   Trying IPv6 address... connected
* Connected to aws.amazon.com (IPv6 address) port 443 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1t
 zlib/1.2.3
> Host: aws.amazon.com
```

使用 Windows 测试网络

如果使用的是 Windows，则使用如下所示的 `ping` 命令来测试您的网络能否通过 IPv6 或 IPv4 访问双栈端点：

```
ping aws.amazon.com
```

如果 ping 通过 IPv6 访问该端点，则该命令会返回 IPv6 地址。

验证客户端是否可以支持 IPv6

我们建议您在切换到使用 internetmonitor.{region}.api.aws 端点前，首先验证客户端是否能够访问其他已启用 IPv6 的 AWS 服务端点。以下步骤介绍如何使用现有的 IPv6 端点进行验证。

此例使用 Linux 和 curl 版本 8.6.0，并使用 Amazon Athena 服务，后者在 api.aws 域具有启用了 IPv6 的端点。

> **Note**
>
> 将您的 AWS 区域切换到客户端所在的区域。在此示例中，我们使用的是美国东部（弗吉尼亚州北部）- us-east-1 端点。

使用以下示例验证客户端是否能够访问启用 IPv6 的 AWS 端点。

1. 使用以下命令验证 Athena 端点是否使用 IPv6 地址进行解析。

```
dig +short AAAA athena.us-east-1.api.aws
2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
2600:1f18:e2f:4e03:4a1e:83b0:8823:4ce5
2600:1f18:e2f:4e04:34c3:6e9a:2b0d:dc79
```

2. 现在，使用以下命令确定客户端网络是否能够使用 IPv6 建立连接：

```
curl --ipv6 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws

remote ip: 2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
response code: 404
```

   如果已识别远程 IP 地址并且响应代码不是 0，则表示已使用 IPv6 成功与端点建立网络连接。

   如果远程 IP 地址为空或响应代码为 0，则表示客户端网络或端点的网络路径仅支持 IPv4。您可使用以下 curl 命令执行此操作：

```
curl -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws
```

```
remote ip: 3.210.103.49
response code: 404
```

如果运行此命令，已识别远程 IP 地址并且响应代码不是 0，则表示已使用 IPv4 成功与端点建立网络连接。

Amazon CloudWatch 网络监测仪如何与 IAM 协同工作

在使用 IAM 管理对 Internet Monitor 的访问之前，您应该了解哪些 IAM 功能可与 Internet Monitor 结合使用。

要查看显示 AWS 服务如何与大多数 IAM 功能结合使用的类似高级视图的表格，请参阅《IAM 用户指南》中的与 IAM 结合使用的 AWS 服务。

可以与 Amazon CloudWatch 网络监测仪结合使用的 IAM 功能

IAM 功能	Internet Monitor 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	否
服务相关角色	是

Internet Monitor 基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Internet Monitor 中基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。

Internet Monitor 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Internet Monitor 操作的列表，请参阅《服务授权参考》https://docs.aws.amazon.com/service-authorization/latest/reference/list_amazoncloudwatchinternetmonitor.html#amazoncloudwatchinternetmonitor-actions-as-permissions中的 Amazon CloudWatch 网络监测仪定义的操作。

Internet Monitor 中的策略操作在操作前使用以下前缀：

```
internetmonitor
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "internetmonitor:action1",
      "internetmonitor:action2"
         ]
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "internetmonitor:Describe*"
```

Internet Monitor 的政策资源

支持策略资源：是

在《服务授权参考》中，您可以看到以下与 Internet Monitor 相关的信息：

- 要查看 Internet Monitor 资源类型及其 ARN 的列表，请参阅 Amazon CloudWatch 网络监测仪定义的资源。
- 要了解您可以使用每个资源的 ARN 指定的操作，请参阅 Amazon CloudWatch 网络监测仪定义的操作。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

Internet Monitor 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

有关 Internet Monitor 条件键的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 网络监测仪的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon CloudWatch 网络监测仪定义的操作。

Internet Monitor 中的 ACL

支持 ACL：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

将 ABAC 与 Internet Monitor 结合使用

支持 ABAC（策略中的标签）：部分支持

Internet Monitor 部分支持策略中的标签。它支持标记一种资源，即监测仪。

要将标签与 Internet Monitor 结合使用，请使用 AWS Command Line Interface 或 AWS 开发工具包。AWS Management Console 不支持标记 Internet Monitor。

要了解有关在一般策略中使用标签的更多信息，请查看以下信息。

基于属性的访问控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证与 Internet Monitor 结合使用

支持临时凭证：是

某些 AWS 服务 在您使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务 与临时凭证配合使用，请参阅 IAM 用户指南中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其它方法登录到 AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录（SSO）链接访问 AWS 时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

Internet Monitor 的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Internet Monitor 的服务角色

支持服务角色：否

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务 委派权限的角色。

Internet Monitor 的服务相关角色

支持服务相关角色：是

服务相关角色是一种与 AWS 服务 相关的服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 AWS 账户 中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

要了解 Internet Monitor 的服务相关角色的更多信息，请参阅 Amazon CloudWatch 网络监测仪的服务相关角色。

有关在 AWS 中创建或管理服务相关角色的大概详细信息，请参阅与 IAM 协同工作的 AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

Amazon CloudWatch 网络监测仪的 AWS 托管策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户托管式策略来进一步减少权限。

您无法更改 AWS 托管式策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务 启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管策略：CloudWatchInternetMonitorServiceRolePolicy

此策略附加到名为 AWSServiceRoleForInternetMonitor 的服务相关角色，以允许网络监测仪访问您账户中的资源，例如 Amazon Virtual Private Cloud 资源或网络负载均衡器，以便您可以在创建监测仪时选择这些资源。有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的服务相关角色。

Amazon CloudWatch 网络监测仪的 IAM 权限

要访问使用 Amazon CloudWatch 网络监测仪中的监测仪和数据的操作，用户必须拥有正确权限。

有关 Amazon CloudWatch 安全性的更多信息，请参阅 适用于 Amazon CloudWatch 的 Identity and Access Management。

Amazon CloudWatch 网络监测仪的只读访问权限

要访问只读操作以使用 Amazon CloudWatch 网络监测仪中的监测仪和数据，用户必须以具有以下权限的用户或角色身份登录：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "internetmonitor:Get*",
                "internetmonitor:List*",
                "internetmonitor:StartQuery",
                "internetmonitor:StopQuery",
                "logs:DescribeLogGroups",
                "logs:GetQueryResults",
                "logs:StartQuery",
                "logs:StopQuery"
                ],
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch 网络监测仪的完全访问权限

要在 Amazon CloudWatch 网络监测仪中创建监测仪，并拥有操作完全访问权限以使用网络监测仪中的监测仪和数据，用户必须以具有以下权限的用户或角色身份登录：

- 具有权限创建与网络监测仪关联的服务相关角色。有关更多信息，请参阅 Amazon CloudWatch 网络监测仪的服务相关角色。
- 具有操作权限，可启用完全访问权限以使用网络监测仪中的监测仪和数据。

> **Note**
>
> 如果创建更为严格的基于身份的权限策略，则采用该策略的用户可能没有完全访问权限以创建与使用网络监测仪中的监测仪和数据。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "internetmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "internetmonitor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor"
        },
        {
            "Action": [
                "ec2:DescribeVpcs",
                "elasticloadbalancing:DescribeLoadBalancers",
                "workspaces:DescribeWorkspaceDirectories",
                "cloudfront:GetDistribution"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon CloudWatch 网络监测仪的服务相关角色

Amazon CloudWatch 网络监测仪使用了 AWS Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Internet Monitor 直接相关。服务相关角色是由 Internet Monitor 预定义的，包含该服务代表您调用其他 AWS 服务所需的所有权限。

Internet Monitor 定义服务相关角色的权限，除非另有定义，否则只有 Internet Monitor 可以担任该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有先删除角色的相关资源，才能删除角色。此限制将保护您的 Internet Monitor 资源，因为您不会无意中删除对这些资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找 Service-linked role（服务相关角色）列中显示为 Yes（是）的服务。选择是和链接，查看该服务的服务相关角色文档。

Internet Monitor 的服务相关角色权限

Internet Monitor 使用名为 AWSServiceRoleForInternetMonitor 的服务相关角色。此角色允许网络监测仪访问您账户中的资源，例如 Amazon 虚拟私有云资源、Amazon CloudFront 分配、Amazon WorkSpaces 目录和网络负载均衡器，以便您可以在创建监测仪时选择这些资源。

此服务相关角色使用托管策略 `CloudWatchInternetMonitorServiceRolePolicy`。

AWSServiceRoleForInternetMonitor 服务相关角色信任以下服务来代入该角色：

- `internetmonitor.amazonaws.com`

要查看此策略的权限，请参阅 AWS 托管策略参考中的 CloudWatchInternetMonitorServiceRolePolicy。

为 Internet Monitor 创建服务相关角色

您无需为 Internet Monitor 手动创建服务相关角色。首次创建监视器时，Internet Monitor 会为您创建 AWSServiceRoleForInternetMonitor。

有关更多信息，请参阅 IAM 用户指南 中的创建服务相关角色。

编辑 Internet Monitor 的服务相关角色

Internet Monitor 在您的账户中创建服务相关角色后，将无法更改角色名称，因为可能有多个实体引用该角色。您可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》 中的编辑服务相关角色。

删除 Internet Monitor 的服务相关角色

如果您不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除服务相关角色的资源，然后才能手动删除它。

当您从 Internet Monitor 的监视器中移除资源并删除监视器后，可以删除服务相关角色 AWSServiceRoleForInternetMonitor。

> **Note**
>
> 如果当您试图删除该角色时 Internet Monitor 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟，然后重试。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForInternetMonitor 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

Internet Monitor 服务相关角色的更新

有关 AWSServiceRoleForInternetMonitor、网络监测仪服务相关角色 AWS 托管策略的更新，请参阅 CloudWatch 对 AWS 托管式策略做出的更新。如需获得 CloudWatch 中托管策略更改的自动提示，请订阅 CloudWatch 历史记录页面上的 RSS 源。

Amazon CloudWatch 网络监测仪的配额

Amazon CloudWatch 网络监测仪具有以下配额。

资源	默认限额
每个区域的监视器数	50

资源	默认限额
每个监视器的资源数	50
已解决的 Internet Monitor 运行状况事件的保留天数	400

使用 Amazon CloudWatch 网络监测仪

您可通过 Amazon CloudWatch 网络监测仪了解将 AWS 托管应用程序连接到本地目标的网络的性能，并可在几分钟内确定网络性能下降的根源。网络监测仪完全由 AWS 托管。因此，您无需安装其他代理即可监控网络性能。您可以快速可视化混合网络连接的丢包率和延迟，设置警报和阈值，然后执行相应操作，从而改善最终用户的网络体验。

网络监测仪适用于想要实时了解网络性能的网络运营商和应用程序开发人员。

主要特征

- 借助网络监测仪，利用持续的实时丢包率和延迟指标对不断变化的混合网络环境进行基准测试。
- 当您使用 AWS Direct Connect 进行连接时，网络监测仪会将 AWS 网络运行状况指标（NHI）写入您的 Amazon CloudWatch 账户，从而快速诊断网络性能下降的情况。您可通过 NHI 指标提供的概率分数确定 AWS 是否曾出现网络性能下降。
- 网络监测仪提供完全托管的代理方法进行监控，因此您无需在 VPC 上或本地安装代理。要开始使用，只需指定 VPC 子网和本地 IP 地址。
- 网络监测仪向 CloudWatch 指标发布指标。您可以创建控制面板来查看您的指标，还可针对特定于应用程序的指标创建可操作阈值和警报。

有关更多详细信息，请参阅the section called “网络监测仪的工作原理”。

网络监测仪术语和组件

- 监测仪：监测仪显示的资源可用于查看应用程序的网络性能和可用性测量数据，以及获取有关应用程序的运行状况事件警报。为应用程序创建监测仪时，您可以将 AWS 托管资源添加为网络源。然后，网络监测仪会创建 AWS 托管资源和目标 IP 地址之间所有可能的探针的列表。
- 探测器：探测器是指从 AWS 托管资源发送到本地目标 IP 地址的流量。对于监测仪内配置的各个探测器，网络监测仪指标会写入您的 CloudWatch 账户。

- AWS 网络源：AWS 网络源是监测仪探针的原始 AWS 来源，该来源将成为其中一个 VPC 中的子网。
- 目标：目标是 AWS 网络源在本地网络中的目标。目标是您的本地 IP 地址、网络协议、端口和网络数据包大小的组合。同时支持 IPv4 和 IPv6 地址。

网络监测仪的限制和要求

- 网络监测仪最多支持四个目标 IP 地址，每个监测仪最多支持 24 个探测器。
- 每个账户每个区域最多可以有 100 个监测仪。
- 监控子网必须由与监控相同的账户拥有。
- 出现 AWS 网络问题时，网络监测仪不提供自动网络失效转移。
- 创建每个探测器都需要付费。有关定价详细信息，请参阅the section called “定价”。

Amazon CloudWatch 网络监测仪的工作原理

网络监测仪由 AWS 完全托管，不需要在受监控的资源上安装代理。为 AWS 托管资源创建监测仪时，AWS 会在后台创建和管理所有基础设施，以测量往返时间和丢包率。由于 AWS 托管所需的配置，因此您无需在 AWS 基础设施中安装或卸载代理，即可快速扩展监控规模。

网络监测仪重点监控来自 AWS 托管资源的流量所采用的路由，而不是广泛监控来自您 AWS 区域 的所有流量。如果您的工作负载分布在多个可用区中，则网络监测仪可以监控来自您各个私有子网的路由。

网络监测仪根据您在创建监测仪时设置的聚合间隔向您的 Amazon CloudWatch 账户发布往返时间和丢包率指标。您还可以使用 CloudWatch 为每个监测仪设置单独的延迟和丢包率阈值。例如，您可以为易受丢包率影响的工作负载创建警报，在平均丢包率高于 0.1% 静态阈值时通知自己。您还可以使用 CloudWatch 异常检测功能对超出所需范围的丢包率或延迟指标发出警报。

可用性和性能测量

网络监测仪定期将处于活动状态的探针从您的 AWS 资源发送到本地目标。创建监测仪时，您可指定以下内容：

- 聚合时间间隔：CloudWatch 收到测量结果的时间（以秒为单位）。即每 30 秒或 60 秒一次。您为监测仪选择的聚合周期适用于该监测仪中的所有探测器。
- 探针协议：支持的协议之一：ICMP 或 TCP。有关更多信息，请参阅 the section called “通信协议”。
- 数据包大小。单个探测器在 AWS 托管资源和目标之间传输的各个数据包大小（以字节为单位）。您可以为监测仪中的每个探针指定不同的数据包大小。

监测仪发布的指标如下：

- 往返时间：该指标以毫秒为单位，是衡量性能的指标。它会记录探针传输到目标 IP 地址以及接收关联响应所花的时间。
- 数据包丢失：该指标会测量已发送数据包总数的百分比，并记录未收到关联响应的已传输探针的数量。没有响应意味着数据包在网络路径上丢失。

支持的通信协议

网络监测仪支持两种探针协议：ICMP 和 TCP。

基于 ICMP 的探针将来自 AWS 托管资源的 ICMP 回显请求传送到目标地址，并期望得到 ICMP 回显回复。网络监测仪使用 ICMP 回显请求和回复消息的相关信息来计算往返时间和丢包率指标。

基于 TCP 的探针将 TCP SYN 数据包从您的 AWS 托管资源传输到目标地址和端口，并预期得到 TCP SYN+ACK 或 RST 数据包响应。网络监测仪使用 TCP SYN 与 TCP SYN+ACK 或 RST 消息的相关信息来计算往返时间和丢包率指标。网络监测仪会定期切换源 TCP 端口，以此扩大网络覆盖范围，从而提高检测到丢包的概率。

AWS 网络运行状况指标

网络监测仪会发布网络运行状况指标（NHI），您可通过该指标了解通过 AWS Direct Connect 连接的目标的网络性能和可用性等信息。NHI 是一种统计指标，用于衡量从 AWS 托管资源（即监测仪部署位置）到 Direct Connect 位置的 AWS 受控网络路径运行状况。

网络监测仪利用异常检测功能来计算网络路径的可用性下降或性能下降情况。

> **Note**
>
> 每次创建新监测仪、添加探针或重新激活探针时，该监测仪的 NHI 将延迟几小时，同时 AWS 会收集数据用于执行异常检测。

为了提供 NHI 指标，网络监测仪将统计相关性应用于 AWS 示例数据集，以及模拟网络路径的流量的丢包率和往返延迟指标。NHI 可以是以下两个值之一：1 或 0。值为 1 表示网络监测仪观测到 AWS 受控网络路径中出现网络性能下降的情况。值为 0 表示网络监测仪未观测到路径中出现任何网络性能下降的情况。观察 NHI 值可以让您更快地发现网络问题。例如，您可以为 NHI 指标设置警报，以在网络路径中发生问题时收到通知。

支持 IPv4 和 IPv6 地址

网络监测仪可通过 IPv4 或 IPv6 网络提供可用性和性能指标，且可监控来自双堆栈 VPC 的 IPv4 或 IPv6 地址。网络监测仪不允许在同一监测仪中同时配置 IPv4 和 IPv6 目标；但您可以为仅限 IPv4 和仅限 IPv6 的目标单独创建监测仪。

区域可用性

目前以下 AWS 区域 可使用网络监测仪：

区域名称	区域
亚太地区（香港）	ap-east-1
亚太地区（孟买）	ap-south-1
亚太地区（首尔）	ap-northeast-2
亚太地区（新加坡）	ap-southeast-1
亚太地区（悉尼）	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
加拿大西部（卡尔加里）	ca-west-1
欧洲（法兰克福）	eu-central-1
欧洲地区（爱尔兰）	eu-west-1
欧洲地区（伦敦）	eu-west-2
欧洲地区（巴黎）	eu-west-3
欧洲地区（斯德哥尔摩）	eu-north-1
中东（巴林）	me-south-1
南美洲（圣保罗）	sa-east-1
美国东部（弗吉尼亚州北部）	us-east-1

区域名称	区域
美国东部（俄亥俄州）	us-east-2
美国西部（加利福尼亚北部）	us-west-1
美国西部（俄勒冈州）	us-west-2

在 Amazon CloudWatch 网络监测仪中创建监测仪

以下各节介绍如何创建监测仪然后添加所需的探针。您可以为每个探针选择一个源子网和最多四个目标 IP 地址，每个监测仪最多支持 24 个探针。您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface 创建监测仪。

主题

- 使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪
- 使用 AWS Command Line Interface 在 Amazon CloudWatch 网络监测仪中创建监测仪

使用控制台在 Amazon CloudWatch 网络监测仪中创建监测仪

以下步骤介绍如何使用 Amazon CloudWatch 控制台创建监测仪。要创建监测仪，您可选择监测仪的源子网，然后最多添加四个目标，每个监测仪最多可以创建 24 个探针。

> **Important**
>
> 这些步骤需一次性完成。任何正在进行的工作都无法保存以待后续处理。

定义监测仪详细信息

创建监测仪的第一步是定义基本详细信息，方法是为监测仪命名和定义聚合周期。（可选）您还可以添加标签。

定义监测仪详细信息

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 选择 Create monitor（创建监控）。

3. 对于监测仪名称，输入监测仪的名称。
4. 对于聚合周期，选择您想要向 CloudWatch 发送指标的频率：30 秒或 60 秒。

> **Note**
>
> 较短的聚合周期有助于更快地检测网络问题；但是，所选聚合周期可能会影响您的账单结构。有关定价的更多信息，请参阅 Amazon CloudWatch 定价页面。

5. （可选）对于标签，添加键和值对以帮助识别此资源，以使您可以搜索或筛选特定信息。
   1. 选择添加新标签。
   2. 输入键名称和关联值。
   3. 要添加新标签，请选择添加新标签。

      选择添加新标签即可添加多个标签，也可选择删除来删除标签。
   4. 如果要将标签与监测仪的探针关联，则请保持选中向监测仪创建的探针添加标签。这样即可向监测仪探针添加标签，这对基于标签的身份验证或计量可能有帮助。
6. 要 the section called “选择源和目标”，请选择下一步。

选择源和目标

网络监测仪中的监测仪将针对 VPC 和网络运行所在区域中的关联子网使用 AWS 源。监测仪目标是本地 IP 地址、网络协议、端口和网络数据包大小的组合。

源和目标的组合就是探针。每个子网最多支持四个探测器，每个监测仪最多支持 24 个探测器。

> **Important**
>
> 这些步骤需一次性完成。任何正在进行的工作都无法保存以待后续处理。

选择源和目标

1. 在 AWS 网络源下，选择要包含在监测仪中的一个或多个子网。要选择 VPC 中的所有子网，请选择该 VPC。或者，选择 VPC 内的特定子网。所选 VPC 和子网均为监测仪源。
2. 在目标 1 中输入本地网络的目标 IP 地址。同时支持 IPv4 和 IPv6 地址。
3. 选择高级设置。

4. 为此客户托管目标选择网络协议。该协议可以是 ICMP 或 TCP。
5. 如果选择 TCP，请输入以下信息：
   1. 输入您的网络用于连接的端口。该端口号必须为介于 1 到 65535 之间的数字。
   2. 输入数据包大小。此即探测器在源和目标之间发送的各个数据包的大小（以字节为单位）。数据包大小必须为介于 56 到 8500 之间的数字。
6. 选择添加目标，向该监测仪添加另一个本地目标。针对您要添加的每个目标重复这些步骤。
7. 完成后选择下一步以确认探测器。

确认探针

确认探针页面会显示所提供探针规格的源和目标所有可能的组合。例如，如果您有六个源子网和四个目标 IP 地址，则总共有 24 种可能的探针组合。

> **Important**
>
> - 这些步骤应在一个会话中完成。您无法保存正在进行的工作以待后续处理。
> - 确认探测器页面不会表明探测器是否有效。我们建议您仔细查看此页面，然后删除任何无效的探针。如果不删除，则您可能需要为无效的探针付费。

确认监测仪探测器

1. 先决条件：the section called “选择源和目标”。
2. 在确认探针页面上，查看源和目标探针组合列表。
3. 选择要从监测仪中删除的所有探针，然后选择删除。

   > **Note**
   >
   > 系统不会提示您确认删除探针。如果您删除探针并想要将其恢复，则必须重新进行设置。您可以按照 the section called “向监测仪添加探测器” 中的步骤操作，向现有监测仪添加探针。

4. 选择下一步以在创建监测仪之前查看详细信息。

审查和创建

最后一步是查看监测仪和监测仪探针的详细信息。此时，您可以更改有关监测仪的任何信息。审查完毕后，更改所有不正确的信息，然后创建监测仪。

创建监测仪后，网络监测仪就会开始跟踪指标，并且您将开始为监测仪中的探针付费。

> **Important**
>
> - 此步骤应在一个会话中完成。您无法保存正在进行的工作以待后续处理。
> - 如果您选择编辑某个部分，则必须从进行编辑的那一刻开始逐步完成创建监测仪的过程。之前的监测仪创建页面会保留您已经输入的信息。

查看并创建监测仪

1. 在查看并创建探测器页面上，为要进行更改的任意部分选择编辑。
2. 在该部分进行任何更改，然后选择下一步。
3. 完成编辑后，选择创建监测仪。

   网络监测仪页面的网络监测仪部分会显示监测仪创建的当前状态。当网络监测仪创建监测仪时，状态为待处理。当状态更改为活动时，您可以通过访问监测仪控制面板查看 CloudWatch 指标。

   有关使用监测仪控制面板的信息，请参阅 the section called “网络监测仪控制面板”。

> **Note**
>
> 新添加的监测仪可能需要几分钟才能开始收集网络指标。

使用 AWS Command Line Interface 在 Amazon CloudWatch 网络监测仪中创建监测仪

您可以不使用 AWS Management Console，而是使用 AWS Command Line Interface 或其他 API 来创建监测仪。

使用命令行创建监测仪

1. 使用 create-monitor 创建监测仪。

2. 使用 create-probe 创建监测仪探针。

使用网络监测仪中的监测仪和探针

您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface，通过监测仪和探针执行以下任何任务。

主题：

- 编辑监测仪
- 删除监测仪
- 激活或停用探针
- 向监测仪添加探测器
- 编辑探测器
- 删除探测器
- 标记和取消标记资源

编辑监测仪

您可以编辑网络监测仪的信息，包括更改名称、设置新的聚合周期，或者添加或删除标签。更改监测仪的信息不会更改任何与之关联的探测器。您可以使用 Amazon CloudWatch 控制台或 CLI 编辑监测仪。

使用控制台编辑监测仪

使用 CloudWatch 控制台编辑监测仪。

使用控制台编辑监测仪

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分，选择要编辑的监测仪。
3. 在监测仪控制面板页面上，选择编辑。
4. 在监测仪名称中输入监测仪的新名称。
5. 在聚合周期中选择要向 CloudWatch 发送指标的频率。有效期为：

- 30 秒
- 60 秒

> **Note**
>
> 较短的聚合周期有助于更快地检测网络问题；但是，所选聚合周期可能会影响您的账单结构。有关定价的更多信息，请参阅 Amazon CloudWatch 定价页面。

6. （可选）在标签部分添加键和值对可进一步帮助识别此项资源，您可以在此基础上搜索或筛选特定信息。您也可以只更改任何当前键的值。

   1. 选择添加新标签。
   2. 输入键名称和关联值。
   3. 要添加新标签，请选择添加新标签。

      选择添加新标签即可添加多个标签，也可选择删除来删除任何标签。
   4. 如果要将标签与监测仪关联，请保持选中向监测仪创建的探测器添加标签。这样即可向监测仪探测器添加标签，如果您使用的是基于标签的身份验证或计量，这种做法很有帮助。
7. 选择 Save changes（保存更改）。

使用 CLI 编辑监测仪

使用 AWS Command Line Interface 编辑网络监测仪中的监测仪。

使用 CLI 编辑监测仪

1. 如果您不知道监测仪名称，则请使用 list-monitors 获取监测仪列表。记下要编辑的监测仪名称。
2. 使用 edit-monitor，然后通过 list-monitors 命令指定监测仪的名称。

删除监测仪

在删除网络监测仪中的监测仪之前，无论监测仪状态如何，必须停用或删除与该监测仪关联的所有探针。停用或删除监测仪后，您不再需要支付监测仪中探针的费用。请注意，已删除的监测仪无法恢复。您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface 删除监测仪。

如果您删除或停用探针，则 CloudWatch 会将其发布的指标保留 15 天。

使用控制台删除监测仪

使用 CloudWatch 控制台删除网络监测仪中的监测仪。

使用控制台删除监测仪

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分，选择要删除的监测仪。
3. 选择操作，然后选择删除。
4. 如果您有任何处于活动状态的监测仪探针，则系统将提示您停用。选择停用探测器。

> **Note**
>
> 选择停用探测器后，您将无法取消或撤消此操作。但已停用的探测器不会从监测仪中删除。如果您愿意，则可以稍后将其重新激活。请参阅 the section called “激活或停用探针”。

5. 在确认字段中，输入 **`confirm`**，然后选择删除。

使用 CLI 删除监测仪

使用 AWS Command Line Interface 删除网络监测仪中的监测仪。

使用 CLI 删除监测仪

1. 要删除监测仪，您需要知道监测仪名称。如果您不知道名称，则请运行 list-monitors 命令获取监测仪列表。记下要删除的监测仪的名称。
2. 验证该监测仪是否包含任何活动的探针。根据上一步的监测仪名称，使用 get-monitor。此操作将返回该监测仪的所有关联探测器的列表。
3. 如果监测仪包含活动的探针，则必须首先将这些探针设置为非活动状态或将其删除。
   - 要将探测器设置为非活动状态，请使用 update-probe，并将状态设置为 `INACTIVE`。
   - 要删除探测器，请使用 delete-probe。
4. 将探针设置为 `INACTIVE` 或删除后，您可以通过运行 delete-monitor 命令来删除监测仪。当您删除监测仪时，不会删除非活动探针。

激活或停用探针

您可以在网络监测仪中激活或停用监测仪中的探针。您可能想要在目前并未使用探针，但将来可能需要再次使用的场景中停用探针。如果停用探针而不是删除探针，则您无需花时间重新设置。您无需支付已停用探测器的费用。

您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface 更改探针的状态。

使用控制台将探针设置为活动或非活动状态

使用 CloudWatch 控制台将探测器设置为活动或非活动状态。

使用控制台将探针设置为活动或非活动状态

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 选择监测仪详细信息选项卡。
3. 在探测器部分，选择要激活或停用的探测器。
4. 选择操作，然后选择激活或停用。

> **Note**
>
> 重新激活探针后，探针会再次开始产生的计费。

使用 CLI 将探针设置为活动或非活动状态

您可以使用 CLI 将探针设置为活动、非活动或停用状态。此命令仅可用于单个探测器。

使用 CLI 将探针设置为活动或非活动状态

1. 如果您不知道监测仪名称，请使用 list-monitors 获取监测仪列表。记下包含要更改状态的探针的监测仪名称。
2. 根据上一步的监测仪名称，使用 get-monitor。此操作将返回该监测仪的所有关联探测器的列表。记下要更改状态的探针的探针 ID。
3. 使用 update-probe 将探针状态设置为 `ACTIVE` 或 `INACTIVE`。

向监测仪添加探测器

您可以向现有监测仪添加探测器。请注意，当您向监测仪添加探针时，账单结构将更新为包含新探针。

使用控制台向监测仪添加探针

使用控制台向监测仪添加探测器

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分，执行以下任一操作：
   - 选择要向其添加探测器的监测仪名称链接。选择监测仪详细信息选项卡，然后在探测器部分选择添加探测器。
   - 选择监测仪复选框，选择操作，然后选择添加探测器。
3. 在添加探测器页面上，执行以下操作：
   1. 在 AWS 网络源下，选择要向监测仪添加的子网。

      > **Note**
      >
      > 一次仅能添加一个探测器，每个监测仪最多可以添加四个探测器。

   2. 输入本地网络的目标 IP 地址。同时支持 IPv4 和 IPv6 地址。
   3. 选择高级设置。
   4. 为目标选择网络协议。该协议可以是 ICMP 或 TCP。
   5. 如果协议为 TCP，请输入以下信息。否则，请跳到下一步：
      - 输入您的网络用于连接的端口。该端口号必须为介于 1 到 65535 之间的数字。
      - 输入数据包大小。此即探测器在源和目标之间发送的各个数据包的大小（以字节为单位）。数据包大小必须为介于 56 到 8500 之间的数字。
4. （可选）在标签部分添加键和值对可进一步帮助识别此项资源，您可以在此基础上搜索或筛选特定信息。
   1. 选择添加新标签。
   2. 输入键名称和关联值。
   3. 要添加新标签，请选择添加新标签。

      选择添加新标签即可添加多个标签，也可选择删除来删除任何标签。

5. 选择添加探测器。

   当探测器处于激活状态时，状态显示为待定。探测器可能需要几分钟才能变为活动状态。

使用 CLI 向监测仪添加探针

使用 AWS Command Line Interface 向监测仪添加探针。使用此命令一次仅能添加单个探测器。

使用命令行或 API 向监测仪添加探测器

1. 如果您不知道监测仪名称，请使用 list-monitors 获取监测仪列表。记下要将探针添加到的监测仪的名称。
2. 使用 create-probe 向监测仪添加探测器。

编辑探测器

无论现有探针处于活动还是非活动状态，您都可以更改该探针的任何信息。您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface 来编辑探针。

使用控制台编辑探针

使用控制台编辑探针

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。

   在名称下，选择监测仪链接，打开监测仪控制面板。
2. 选择监测仪详细信息选项卡。
3. 在探针部分，选择要编辑的探针的链接。
4. 在探针详情页上，选择编辑。
5. 在编辑探针页面上，输入探针的新目标 IP 地址。同时支持 IPv4 和 IPv6 地址。
6. 选择高级设置。
7. 依次选择网络协议、ICMP 或 TCP。
8. 如果协议为 TCP，则请输入以下信息：

   - 输入您的网络用于连接的端口。该端口号必须为介于 1 到 65535 之间的数字。
   - 输入数据包大小。此即探测器在源和目标之间发送的各个数据包的大小（以字节为单位）。数据包大小必须为介于 56 到 8500 之间的数字。

9. （可选）添加、更改或删除探针的标签。
10. 选择 Save changes（保存更改）。

使用 CLI 编辑探针

您可以使用 AWS Command Line Interface 编辑监测仪探针。使用此命令一次只能编辑一个探针。

使用 CLI 编辑探针

1. 使用 list-monitors 获取监测仪列表，并记下包含要编辑的探针的监测仪名称。
2. 将 get-monitor 与监测仪名称一起使用。此操作将返回该监测仪关联探针的列表。记下要编辑的一个或多个探针的探针 ID。
3. 使用 update-probe 更改探测器信息。

删除探测器

如果您确认以后不再需要使用探针，则可将其删除而非停用。您无法恢复已删除的探针；相反地，您必须重新创建该探针。删除探针后，该探针将停止计费。您可以使用 Amazon CloudWatch 控制台或 AWS Command Line Interface 删除探针。

使用控制台删除探针

使用控制台删除探测器

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分的名称下，选择监测仪链接以打开监测仪控制面板。
3. 选择监测仪详细信息选项卡。
4. 选择监测仪复选框，选择操作，然后选择删除。
5. 在删除探针对话框中，执行以下操作：
6. 选择删除，确认您要删除探测器。

   探测器部分的探测器状态显示为正在删除。删除后，探测器将从探测器部分删除。

使用 CLI 删除探针

使用 AWS Command Line Interface 删除探针。您只能使用此命令删除单个探针。

使用 CLI 删除探针

1. 如果您不知道监测仪名称，请使用 list-monitors 获取监测仪列表。记下包含要删除的探测器的监测仪名称
2. 根据上一步的监测仪名称，使用 get-monitor。此操作将返回该监测仪的所有关联探测器的列表。记下要删除的任何探测器的探测器 ID。
3. 使用 delete-probe。

标记和取消标记资源

您可以使用 CLI 来处理资源标记。

使用 CLI 更新监测仪标签

- 要列出资源标签，请使用 list-tags-for-resources。
- 要标记资源，请使用 tag-resource。
- 要取消标记资源，请使用 untag-resource。

网络监测仪控制面板

您可以使用 Amazon CloudWatch 网络监测仪控制面板查看 AWS 网络运行状况、探测器往返时间和丢包率。您可以查看两个监测仪和单个探测器的这些指标。

网络监测仪控制面板

- 监测仪控制面板
- 探测器控制面板

探测器警报

您可以根据 Amazon CloudWatch 网络监测仪指标创建 Amazon CloudWatch 警报，也可为其他 Amazon CloudWatch 指标创建警报。您创建的任何警报在触发时，都将在网络监视器控制面板的监控详细信息部分，于该探测器的状态列中显示。状态将为正常或在警报中。如果探测器未显示任何状态，则表示没有为该探测器创建警报。

例如，您可以根据网络监测仪指标 `PacketLoss` 创建警报，并将其配置为在指标高于选定值时发送通知。您可以遵循与其他 CloudWatch 指标相同的指南，为网络监测仪指标配置警报。

为网络监测仪创建 CloudWatch 警报时，可以使用 AWS/NetworkMonitor 下方的以下指标。

- HealthIndicator
- PacketLoss
- RTT（往返时间）

有关创建网络监测仪警报的步骤，请参阅 the section called “根据静态阈值创建告警”。

设置指标的时间范围

两个控制面板上的指标和事件的默认时间为两个小时，从当前时间开始计算。您可以将默认设置更改为以下任一预设值：

- 1h：一小时
- 2h：两小时
- 1d：一天
- 1w：一周

您还可以设置自定义时间范围。选择自定义，选择绝对或相对时间，然后将时间范围设置为自选时间。根据 CloudWatch 的默认设置，相对时间仅支持从当天日期前推 15 天。

此外，您可以根据 UTC 时区或本地时区选择图表所示时间。

监测仪控制面板

您可以使用 Amazon CloudWatch 网络监测仪控制面板查看 AWS 网络运行状况、探测器往返时间和丢包率。网络监测仪为监测仪和探测器配备了控制面板。

访问监测仪控制面板

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分，选择名称链接，打开监测仪控制面板。

概述

“概览”页面将显示监测仪的以下信息：

- AWS 网络运行状况：AWS 网络运行状况仅显示 AWS 网络的整体运行状况。状态将为正常或性能下降。状态为正常表示网络监测仪未发现 AWS 网络存在任何问题。状态为性能下降表示网络监测仪发现 AWS 网络存在问题。此部分的状态栏显示默认时间（一小时）内的网络状态。将鼠标悬停在状态栏的任意位置上方即可查看更多详细信息。
- 探测器流量摘要：显示监测仪中源 AWS 子网与目标 IP 地址之间流量的当前状态。探测器流量摘要显示以下内容：
  - 处于警报状态的探测器：此数字表示处于性能下降状态的探测器数量。如果您设置为警报的指标被触发，就会相应触发警报。有关网络监视器指标警报的信息，请参阅 the section called “探测器警报”。
  - 丢包率：从源子网到目标 IP 地址途中丢失的数据包数量。该值使用已发送数据包总数的百分比表示。
  - 往返时间：源子网的数据包抵达目标 IP 地址然后再次返回所花费的时间（以毫秒为单位）。

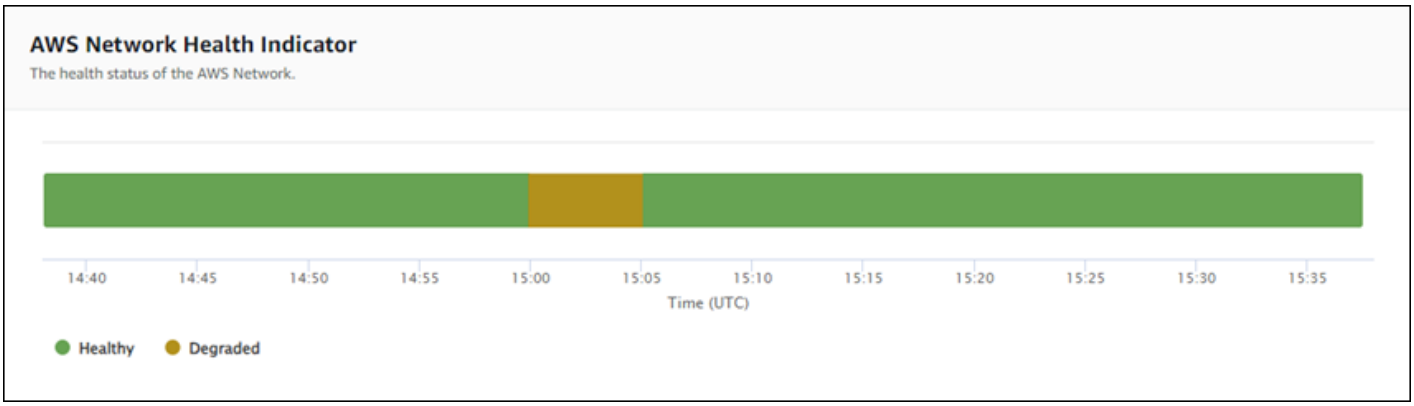
数据用交互式图形表示，可供您查看详细信息。

默认情况下，显示根据当前日期和时间计算出的两小时时间范围内的数据。但您可以根据您的需求更改范围。有关更多信息，请参阅 the section called “设置指标的时间范围”。

跟踪指标

网络监测仪控制面板以图形形式显示您的监测仪和探测器。可供使用图形如下：

- AWS 网络运行状况指标：表示 AWS 网络在指定时间段内的运行状况。状态将为正常或性能下降。在以下示例中，您将看到 AWS 网络在 15:00 UTC 至 15:05 UTC 期间处于性能下降状态。15:05 之后，网络恢复正常状态。将鼠标悬停在图形的任何部分上方即可查看更多详细信息。



> **Note**
>
> 网络运行状况指标并不指示探测器的运行状况，而仅指示 AWS 网络的运行状况。

- 丢包率：此图形中仅显示一条线，显示监测仪中各个探测器的丢包百分比。页面底部的图例显示监测仪中的各个探测器，探测器经过颜色编码，确保唯一性。将鼠标悬停在此图表中的探测器上方即可显示源子网、目标 IP 和丢包百分比。在以下示例中，为从子网探测到 IP 地址 127.0.0.1 的探测器设置了丢包警报。警报在超出探测器的丢包率阈值时触发。将鼠标悬停在图形上方即可显示探测器源位置和目标位置，并且该探测器在 11 月 21 日 02:41:30 的丢包率为 30.97%。

- 往返时间：此图形中的每个探测器均用一条线表示，显示各个探测器的往返时间。页面底部的图例显示监测仪中的各个探测器，探测器经过颜色编码，确保唯一性。将鼠标悬停在此图表中的探测器上方即可显示源子网、目标 IP 地址和往返时间。以下示例显示，11 月 21 日星期二 21:45:30，探测器从子网到 IP 地址 127.0.0.1 的往返时间为 0.075 秒。

监测仪详细信息

监测仪详细信息页面显示有关监测仪（包括探测器）的详细信息。您可以在此页面管理标签或添加探测器。此页面分为以下三个部分：

- 监测仪详细信息：此页面提供有关监测仪的详细信息。无法编辑此部分的信息。但是，您可以选择角色名称链接，查看网络监测仪服务相关角色的详细信息。
- 探测器：此部分显示监测仪所有关联探测器的列表。选择 VPC 或子网 ID 链接，在 Amazon VPC 控制台中打开 VPC 或子网的详细信息。您也可以修改探测器，包括激活或停用探测器。有关更多信息，请参阅 the section called “使用监测仪和探测器”。

  探测器部分会显示有关为该监视器设置的每个探测器的信息，包括探测器 ID、VPC ID、子网 ID、IP 地址、协议，以及该探测器的状态是活动还是非活动。如果为探测器设置了警报，则会显示该警报的当前状态。正常表示没有任何指标事件触发任何警报；在警报中状态表示您在 CloudWatch 中设置的某个指标触发了警报。如果探测器未显示任何状态，则表示未设置 CloudWatch 警报。要了解您可以创建的网络监视器探测警报类型，请参阅 the section called “探测器警报”。
- 标签：查看监测仪的当前标签。您可以通过选择管理标签添加或删除标签。编辑探测器页面随即打开。有关编辑标签的更多信息，请参阅 the section called “编辑监测仪”。

探测器控制面板

您可以使用 Amazon CloudWatch 网络监测仪控制面板来查看 AWS 网络运行状况，以及有关特定探测器的特定往返时间和丢包率的信息。有两个探测器控制面板，即概览和探测器详细信息。

您可以创建 CloudWatch 警报，以设置丢包率和往返时间指标阈值。当某个指标达到阈值时，CloudWatch 警报会通知您。有关创建探测器警报的信息，请参阅 the section called “探测器警报”。

访问探测器控制面板

1. 点击 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台，然后在网络监控下选择网络监测仪。
2. 在网络监测仪部分，选择名称链接，打开监测仪控制面板。
3. 选择 ID 链接，查看该探测器的控制面板。

概述

概览页面显示探测器的以下信息：

- AWS 网络运行状况详细信息：仅提供 AWS 网络的整体运行状况。状态将为正常或性能下降。性能下降状态表明 AWS 网络存在问题，并不表明您的探测器是否存在问题。
- 丢包率：探测器从源子网到目标 IP 地址途中丢失的数据包数量。
- 往返时间：源子网的数据包抵达目标 IP 地址然后再次返回所花费的时间（以毫秒为单位）。

探测器详细信息

探测器详细信息页面显示有关探测器的详细信息。您可以在此页面编辑探测器。有关更多信息，请参阅 the section called “使用监测仪和探测器”。

- 探测器详细信息：此页面提供有关探测器的一般信息。无法编辑此部分的信息。
- 探测器源位置和目标位置：此部分显示有关探测器的详细信息。选择 VPC 或子网 ID 链接，在 Amazon VPC 控制台中打开 VPC 或子网的详细信息。您也可以修改探测器，包括激活或停用探测器。
- 标签：查看监测仪的当前标签。您可以通过选择管理标签添加或删除标签。编辑探测器页面随即打开。有关编辑标签的更多信息，请参阅 the section called “编辑探测器”。

网络监测仪限额

以下是网络监测仪限额：

限额	默认	可调整
每个账户每个 AWS 区域 的最大监测仪数量	100	是
每个监测仪的最大探测器数量	24	是
每个子网每个监测仪的最大探测器数量	4	是

网络监测仪中的数据安全和数据保护

AWS 的云安全性的优先级最高。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 的客户，您也可以从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 Amazon CloudWatch 网络监测仪的合规性计划，请参阅按合规性计划提供的范围内 AWS 服务。
- 云中的安全性：您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 CloudWatch 网络监测仪时应用责任共担模式。以下主题说明了如何配置 CloudWatch 网络监测仪以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护您的 CloudWatch 网络监测仪资源。

主题

- Amazon CloudWatch 网络监测仪中的数据保护
- Amazon CloudWatch 网络监测仪中的基础设施安全性

Amazon CloudWatch 网络监测仪中的数据保护

AWS 责任共担模式适用于 Amazon CloudWatch 网络监测仪中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础架构。您负责维护对托管在此基础架构上的内容的控制。

您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management（IAM）设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日记账记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅美国联邦信息处理标准（FIPS）140-3。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。如您使用控制台、API、AWS CLI 或 AWS SDK，使用 CloudWatch 网络监测仪或其他 AWS 服务 时，以上建议同样适用。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

Amazon CloudWatch 网络监测仪中的基础设施安全性

作为一项托管式服务，Amazon CloudWatch 网络监测仪由 Amazon Web Services: Overview of Security Processes 白皮书中所述的 AWS 全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用，来通过网络访问 CloudWatch 网络监测仪。客户端必须支持传输层安全性协议 (TLS) 1.0 或更高版本。建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 DHE (Ephemeral Diffie-Hellman) 或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

适用于 Amazon CloudWatch 网络监测仪的 Identity and Access Management

AWS Identity and Access Management（IAM）是一项 AWS 服务，可帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 CloudWatch 网络监测仪资源的人员。IAM 是一项可以免费使用的 AWS 服务。利用 IAM 的功能，可在不共享您的安全凭证的情况下允许其他用户、服务和应用程序完全使用或受限使用您的 AWS 资源。

默认情况下，IAM 用户没有创建、查看或修改 AWS 资源的权限。要允许 IAM 用户访问资源（例如全球网络）并执行任务，您必须：

- 创建授予用户使用所需特定资源和 API 操作的权限的 IAM 策略
- 将策略附加到 IAM 用户或用户所属的组

在将策略附加到一个用户或一组用户时，它会授权或拒绝用户对指定资源执行指定任务。

条件键

在 `Condition` 元素（或条件块）中，您可以指定语句生效的条件。条件元素为可选元素。您可以构建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的 IAM JSON 策略元素：条件运算符。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 AWS 使用逻辑 `AND` 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 `OR` 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。

您可以将标签附加到 CloudWatch 网络监测仪资源或将请求中的标签传递到 Cloud WAN。要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的 IAM JSON 策略元素：条件。

要查看所有 AWS 全局条件键，请参阅《AWS Identity and Access Management 用户指南》中的 AWS 全局条件上下文密钥。

标记核心网络资源

标签是您或 AWS 分配给 AWS 资源的元数据标签。每个标签均包含一个键和一个值。对于您分配的标签，需要定义键和值。例如，您可以将键定义为 `purpose`，将一个资源的值定义为 `test`。标签可帮助您：

- 标识和整理您的 AWS 资源。许多 AWS 服务支持标记，因此，您可以将同一标签分配给来不同服务的资源，以指示这些资源是相关的。
- 控制对 AWS 资源的访问。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的 使用标签控制对 AWS 资源的访问。

Amazon CloudWatch 网络监测仪如何与 IAM 协同工作

在使用 IAM 管理对 CloudWatch 网络监测仪的访问之前，您应该了解哪些 IAM 功能可与 CloudWatch 网络监测仪结合使用。

可以与 Amazon CloudWatch 网络监测仪结合使用的 IAM 功能

IAM 功能	CloudWatch 网络监测仪支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是
主体权限	是
服务角色	否
服务相关角色	是

要大致了解 CloudWatch 网络监测仪和其他 AWS 服务如何与大多数 IAM 功能结合使用，请参阅《IAM 用户指南》中的使用 IAM 的 AWS 服务。

Amazon CloudWatch 网络监测仪基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

CloudWatch 网络监测仪基于身份的策略示例

要查看 CloudWatch 网络监测仪基于身份的策略示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

CloudWatch 网络监测仪中基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当主体和资源处于不同的 AWS 账户 中时，则信任账户中的 IAM 管理员还必须授予主体实体（用户或角色）对资源的访问权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

CloudWatch 网络监测仪的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 CloudWatch 网络监测仪操作列表，请参阅《服务授权参考》中的 Amazon CloudWatch 网络监测仪定义的操作。

CloudWatch 网络监测仪中的策略操作在操作前使用以下前缀：

```
networkmonitor
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "networkmonitor:action1",
      "networkmonitor:action2"
         ]
```

要查看 CloudWatch 网络监测仪基于身份的策略示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

CloudWatch 网络监测仪的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 `Resource` 或 `NotResource` 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

有关 CloudWatch 网络监测仪资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 网络监测仪定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon CloudWatch 网络监测仪定义的操作。

CloudWatch 网络监测仪的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

有关 CloudWatch 网络监测仪条件键的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 网络监测仪的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon CloudWatch 网络监测仪定义的操作。

CloudWatch 网络监测仪中的 ACL

支持 ACL：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC 与 CloudWatch 网络监测仪

支持 ABAC（策略中的标签）：部分支持

基于属性的访问控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证用于 CloudWatch 网络监测仪

支持临时凭证：是

某些 AWS 服务 在您使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务 与临时凭证配合使用，请参阅 IAM 用户指南中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其它方法登录到 AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录（SSO）链接访问 AWS 时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

CloudWatch 网络监测仪的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

CloudWatch 网络监测仪的服务角色

支持服务角色：否

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务 委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 CloudWatch 网络监测仪的功能。仅当 CloudWatch 网络监测仪提供相关指导时才编辑服务角色。

将服务相关角色用于 CloudWatch 网络监测仪

支持服务相关角色：是

服务相关角色是一种与 AWS 服务 相关的服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 AWS 账户 中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的 AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

CloudWatch 网络监测仪基于身份的策略示例

默认情况下，用户和角色没有创建或修改 CloudWatch 网络监测仪资源的权限。他们也无法使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM 策略。

有关 CloudWatch 网络监测仪定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅《服务授权参考》中的 Amazon CloudWatch 网络监测仪的操作、资源和条件键。

主题

- 策略最佳实践
- 使用 CloudWatch 网络监测仪控制台
- 允许用户查看他们自己的权限
- CloudWatch 网络监测仪身份和访问问题排查

策略最佳实践

基于身份的策略可确定某个人是否可以创建、访问或删除您账户中的 CloudWatch 网络监测仪资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- AWS 托管策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 AWS 托管策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。我们建议通过定义特定于您的使用场景的 AWS 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略或工作职能的 AWS 托管策略。
- 应用最低权限 – 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定（AWS 服务例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证（MFA）– 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 CloudWatch 网络监测仪控制台

要访问 Amazon CloudWatch 网络监测仪控制台，您必须拥有最低权限。这些权限必须允许您列出和查看有关您 AWS 账户 中的 CloudWatch 网络监测仪资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于只需要调用 AWS CLI 或 AWS API 的用户，无需为其提供最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍可使用 CloudWatch 网络监测仪控制台，请同时将 CloudWatch 网络监测仪 *ConsoleAccess* 或 *ReadOnly* AWS 托管策略添加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

CloudWatch 网络监测仪身份和访问问题排查

使用以下信息来帮助您诊断和修复在使用 CloudWatch 网络监测仪和 IAM 时可能遇到的常见问题。

主题

- 我无权在 CloudWatch 网络监测仪中执行操作
- 我无权执行 iam:PassRole
- 我希望允许我的 AWS 账户 以外的人访问我的 CloudWatch 网络监测仪资源

我无权在 CloudWatch 网络监测仪中执行操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 networkmonitor:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 networkmonitor:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 networkmonitor:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam:PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略才能将角色传递给 CloudWatch 网络监测仪。

有些 AWS 服务 允许将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 CloudWatch 网络监测仪中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我希望允许我的 AWS 账户 以外的人访问我的 CloudWatch 网络监测仪资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 CloudWatch 网络监测仪是否支持这些功能，请参阅 Amazon CloudWatch 如何与 IAM 协同工作。
- 要了解如何为您拥有的 AWS 账户中的资源提供访问权限，请参阅《IAM 用户指南》中的为您拥有的另一个 AWS 账户中的 IAM 用户提供访问权限。
- 要了解如何为第三方 AWS 账户 提供您的资源的访问权限，请参阅 IAM 用户指南中的为第三方拥有的 AWS 账户 提供访问权限。
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

CloudWatch 网络监测仪的 AWS 托管策略

要向用户、组和角色添加权限，与自己编写策略相比，使用 AWS 托管策略更简单。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

AWS 服务负责维护和更新 AWS 管理型策略。您无法更改 AWS 管理型策略中的权限。服务偶尔会向 AWS 管理型策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 管理型策略。服务不会从 AWS 管理型策略中删除权限，因此策略更新不会破坏您的现有权限。

此外，AWS 还支持跨多种服务的工作职能的托管策略。例如，`ReadOnlyAccess` AWS 托管式策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动新特征时，AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 IAM 用户指南中的适用于工作职能的 AWS 管理型策略。

AWS 托管策略：CloudWatchNetworkMonitorServiceRolePolicy

`CloudWatchNetworkMonitorServiceRolePolicy` 附加到服务相关角色，该角色允许服务代表您执行操作并访问 CloudWatch 网络监测仪关联资源。您无法将此策略附加到 IAM 身份。有关更多信息，请参阅 the section called “服务相关角色”。

CloudWatch 网络监控的 AWS 托管策略更新

查看有关 CloudWatch 网络监控的 AWS 托管策略更新的详细信息（从该服务自 2023 年 11 月开始跟踪这些更改开始）。

更改	描述	日期
CloudWatchNetworkMonitorServiceRolePolicy：新策略。	新策略已添加到 CloudWatch 网络监测仪。	2023 年 11 月 27 日
the section called “AWSServiceRoleForNetworkMonitor ”新角色。	新角色已添加到 CloudWatch 网络监测仪。	2023 年 11 月 27 日

CloudWatch 网络监测仪的 IAM 权限

要使用 Amazon CloudWatch 网络监测仪，用户必须拥有正确的权限。

有关 Amazon CloudWatch 安全性的更多信息，请参阅 适用于 Amazon CloudWatch 的 Identity and Access Management。

查看监测仪所需的权限

要在 AWS Management Console 中查看 Amazon CloudWatch 网络监测仪的监测仪，您必须以具有以下权限的用户或角色身份登录：

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "networkmonitor:Get*",
```

```
                "networkmonitor:List*"
                ],
            "Resource": "*"
        }
    ]
}
```

创建监测仪所需的权限

要在 Amazon CloudWatch 网络监测仪中创建监测仪，用户必须有权创建与网络监测仪关联的服务相关角色。要了解有关服务相关角色的更多信息，请参阅 将服务相关角色用于 CloudWatch 网络监测仪。

要在 AWS Management Console 中创建 Amazon CloudWatch 网络监测仪的监测仪，您必须以具有以下策略包含的权限的用户或角色身份登录。

> **Note**
>
> 如果您创建更为严格的基于身份的权限策略，则采用该政策的用户将无法创建监测仪。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "networkmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "networkmonitor.amazonaws.com"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:GetRole",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor"
        },
        {
            "Action": [
                "ec2:CreateSecurityGroup",
                "ec2:CreateNetworkInterface",
                "ec2:CreateTags"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

将服务相关角色用于 CloudWatch 网络监测仪

Amazon CloudWatch 网络监测仪通过以下服务相关角色获取代表您调用其他 AWS 服务所需的权限：

- AWSServiceRoleForNetworkMonitor

AWSServiceRoleForNetworkMonitor

CloudWatch 网络监控使用名为 AWSServiceRoleForNetworkMonitor 的服务相关角色更新和管理 CloudWatch 网络监测仪。

AWSServiceRoleForNetworkMonitor 服务相关角色仅信任以下服务来担任该角色：

- networkmonitor.amazonaws.com

CloudWatchNetworkMonitorServiceRolePolicy 附加到服务相关角色，授予服务访问您账户中的 VPC 和 EC2 资源以及管理已创建网络监测仪的访问权限。

权限分组

策略分组为以下权限集：

- **cloudwatch**：允许服务主体向 CloudWatch 资源发布网络监控指标。
- **ec2**：允许服务主体描述您账户中的 VPC 和子网，以创建或更新监测仪和探测器。此权限集还允许服务主体创建、修改和删除安全组、网络接口及其关联权限，以配置监测仪或探测器，来向您的端点发送监控流量。

有关策略的更多信息，请参阅 the section called “AWS 托管策略”。

CloudWatchNetworkMonitorServiceRolePolicy 如下：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "PublishCw",
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": "AWS/NetworkMonitor"
    }
   }
  },
  {
   "Sid": "DescribeAny",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeNetworkInterfaceAttribute",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfacePermissions",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups"
   ],
   "Resource": "*"
  },
  {
   "Sid": "DeleteModifyEc2Resources",
   "Effect": "Allow",
   "Action": [
    "ec2:AuthorizeSecurityGroupEgress",
    "ec2:CreateNetworkInterfacePermission",
```

```
    "ec2:DeleteNetworkInterfacePermission",
    "ec2:RevokeSecurityGroupEgress",
    "ec2:ModifyNetworkInterfaceAttribute",
    "ec2:DeleteNetworkInterface",
    "ec2:DeleteSecurityGroup"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:network-interface/*",
    "arn:aws:ec2:*:*:security-group/*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceTag/ManagedByCloudWatchNetworkMonitor": "true"
    }
   }
  }
 ]
}
```

创建服务相关角色

AWSServiceRoleForNetworkMonitor

您无需手动创建 AWSServiceRoleForNetworkMonitor 角色。

- CloudWatch 网络监测仪会在您首次创建网络监测仪时创建 AWSServiceRoleForNetworkMonitor 角色。该角色将应用于您后续创建的任何监测仪。

要代表您创建服务相关角色，您必须具有所需权限。有关更多信息，请参阅 IAM 用户指南 中的服务相关角色权限。

编辑服务相关角色

您可以使用 IAM 编辑 AWSServiceRoleForNetworkMonitor 描述。有关更多信息，请参阅 IAM 用户指南 中的编辑服务相关角色。

删除服务相关角色

如果您不再需要使用 CloudWatch 网络监测仪，我们建议您删除 AWSServiceRoleForNetworkMonitor 角色。

您只有在删除网络监测仪后，才能删除服务相关角色。有关删除网络监测仪的信息，请参阅 Delete a network monitor。

您可以使用 IAM 控制台、IAM CLI 或 IAM API 删除服务相关角色。有关更多信息，请参阅 IAM 用户指南 中的删除服务相关角色。

删除 AWSServiceRoleForNetworkMonitor 后，CloudWatch 网络监测仪将在创建新的监测仪时再次创建角色。

CloudWatch 网络监测仪服务相关角色支持的区域

CloudWatch 网络监测仪支持在提供该服务的所有 AWS 区域 使用服务相关角色。有关更多信息，请参阅 AWS 一般参考 中的 AWS 端点。

删除服务相关角色

如果您不再需要使用 CloudWatch 网络监测仪，我们建议您删除 AWSServiceRoleForNetworkMonitor 角色。

您只有在删除网络监测仪后，才能删除服务相关角色。有关删除网络监测仪的信息，请参阅 Delete a network monitor。

您可以使用 IAM 控制台、IAM CLI 或 IAM API 删除服务相关角色。有关更多信息，请参阅 IAM 用户指南 中的删除服务相关角色。

删除 AWSServiceRoleForNetworkMonitor 后，CloudWatch 网络监测仪将在创建新的监测仪时再次创建角色。

Amazon CloudWatch 网络监测仪的定价

使用 Amazon CloudWatch 网络监测仪无需预付费用或长期订阅。网络监测仪的定价包括以下两个组成部分：

- 各项受监控资源的小时费率
- CloudWatch 指标费用

创建网络监测仪时，可以将资源与之关联以进行监控。对于网络监测仪，这些资源是 Amazon Virtual Private Cloud（VPC）中的子网。您可以利用各项受监控资源在 VPC 各子网与四个目标之间创建最多四个探测器。为了帮助您控制账单成本，您可以通过减少所监控资源的数量来调整子网覆盖范围和本地 IP 地址目标覆盖范围。

有关定价的更多信息，请参阅 Amazon CloudWatch 定价页面。

基础设施监控

本部分中的主题介绍了 CloudWatch 的功能，这些功能可以帮助您获得对 AWS 资源的操作了解。

主题

- Container Insights
- Lambda Insights
- 使用 Contributor Insights 分析高基数数据
- Amazon CloudWatch Application Insights
- 使用 CloudWatch 控制台中的资源运行状况视图

Container Insights

使用 CloudWatch Container Insights 可以从容器化应用程序和微服务中收集、聚合和汇总指标与日志。Container Insights 可用于 Amazon EC2 上的 Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS) 和 Kubernetes 平台。Container Insights 支持从部署在 AWS Fargate 上的集群中收集针对 Amazon ECS 和 Amazon EKS 的指标。

CloudWatch 会自动收集许多资源（例如 CPU、内存、磁盘和网络）的指标。Container Insights 还提供诊断信息（如容器重新启动失败），以帮助您查明问题并快速解决问题。您还可以设置 Container Insights 收集的指标的 CloudWatch 告警。

Container Insights 使用嵌入式指标格式将数据收集为性能日志事件。这些性能日志事件是使用结构化 JSON 架构的条目，该架构允许批量提取和存储高基数数据。从该数据中，CloudWatch 在集群、节点、pod、任务和服务级别创建聚合指标以作为 CloudWatch 指标。Container Insights 收集的指标可在 CloudWatch 自动控制面板中使用，也可在 CloudWatch 控制台的指标部分查看。容器任务运行一段时间后，指标才可见。

当您部署 Container Insights 时，它会自动为性能日志事件创建日志组。您无需自行创建该日志组。

为帮助您管理 Container Insights 成本，CloudWatch 不会根据日志数据自动创建所有可能的指标。但是，您可以通过使用 CloudWatch Logs Insights 分析原始性能日志事件来查看其他指标和其他精细级别。

在 Container Insights 的原始版本中，收集的指标和摄取的日志将作为自定义指标收费。借助针对 Amazon EKS 增强了可观测性的 Container Insights，Container Insights 指标和日志按每次观测收

费，而不是按存储的指标或摄取的日志收费。有关 CloudWatch 定价的更多信息，请参阅 Amazon CloudWatch 定价。

在 Amazon EKS 和 Kubernetes 上，Container Insights 使用 CloudWatch 代理的容器化版本在集群中查找所有运行的容器。然后，它在每个性能堆栈层收集性能数据。

Container Insights 支持使用 AWS KMS key 对其收集的日志和指标进行加密。要启用此加密，必须手动为接收 Container Insights 数据的日志组启用 AWS KMS 加密。这将导致 Container Insights 使用提供的 KMS 密钥加密这些数据。仅支持对称密钥。请勿使用非对称 KMS 密钥加密日志组。

有关更多信息，请参阅使用 AWS KMS 加密 CloudWatch Logs 中的日志数据。

针对 Amazon EKS 增强了可观测性的 Container Insights

2023 年 11 月 6 日，发布了 Container Insights 的新版本。此版本支持针对在 Amazon EC2 上运行的 Amazon EKS 集群增强的可观测性，并且可以从这些集群收集更详细的指标。安装后，它会自动为您的 Amazon EKS 集群收集详细的基础设施遥测数据和容器日志。然后，您可以使用精选的、可立即使用的控制面板，深入研究应用程序和基础设施的遥测数据。

针对 Amazon EKS 增强了可观测性的 Container Insights 可收集容器级别的精细运行状况、性能和状态指标，以及控制面板指标。有关收集的其他指标和维度的更多信息，请参阅 Amazon EKS 和 Kubernetes Container Insights 指标。

如果您在 2023 年 11 月 6 日之后在 Amazon EC2 上的 Amazon EKS 集群上使用 CloudWatch 代理安装了 Container Insights，便拥有针对 Amazon EKS 增强了可观测性的 Container Insights。否则，您可以按照 升级到针对 Amazon EKS 增强了可观测性的 Container Insights 中的说明将 Amazon EKS 集群升级到此新版本。

Container Insights 支持 CloudWatch 跨账户可观测性。您可以使用一个监控账户对一个区域内跨多个 AWS 账户的应用程序进行监控和问题排查。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

针对 Amazon EKS 增强了可观测性的 Container Insights 还支持 Windows Worker 节点。

Fargate 不支持针对 Amazon EKS 增强了可观测性的 Container Insights。

> **Note**
>
> 通过导航到 Container Insights 控制台，您可以了解您的集群是否可以升级到针对 Amazon EKS 增强了可观测性的 Container Insights。为此，请在 CloudWatch 控制台的导航窗格中依次选择 Insights、Container Insights。在 Container Insights 控制台中，横幅会告知您是否有任何可以升级的 Amazon EKS 集群，并会链接到升级页面。

支持的平台

Container Insights 可用于 Amazon EC2 实例上的 Amazon Elastic Container Service、Amazon Elastic Kubernetes Service 和 Kubernetes 平台。

- 对于 Amazon ECS，Container Insights 同时在 Linux 和 Windows Server 实例上的集群、任务和服务级别收集指标。它在 Linux 实例上只收集实例级别的指标。

  对于 Amazon ECS，网络指标仅适用于 `bridge` 网络模式和 `awsvpc` 网络模式下的容器。它们不适用于 `host` 网络模式下的容器。
- 对于 Amazon Elastic Kubernetes Service 和 Amazon EC2 实例上的 Kubernetes 平台，仅在 Linux 实例上支持 Container Insights。

CloudWatch 代理容器镜像

亚马逊在 Amazon Elastic Container Registry 上提供 CloudWatch 代理容器镜像。有关更多信息，请参阅 Amazon ECR 上的 cloudwatch-agent。

设置 Container Insights

对于 Amazon ECS、Amazon EKS 和 Kubernetes，Container Insights 设置过程不同。

主题

- 在 Amazon ECS 上设置 Container Insights
- 在 Amazon EKS 和 Kubernetes 上设置 Container Insights

在 Amazon ECS 上设置 Container Insights

您可以使用以下一个或两个选项在 Amazon ECS 集群上启用 Container Insights：

- 使用 AWS Management Console或 AWS CLI 开始收集集群级别、任务级别和服务级别指标。
- 将 CloudWatch 代理部署为进程守护程序服务，来开始采集 Amazon EC2 实例上托管的集群的实例级别指标。

主题

- 在 Amazon ECS 上针对集群级别和服务级别指标设置 Container Insights
- 使用 AWS Distro for OpenTelemetry 在 Amazon ECS 上设置 Container Insights

- 部署 CloudWatch 代理以收集 Amazon ECS 上的 EC2 实例级别指标
- 部署 AWS Distro for OpenTelemetry 以收集 Amazon ECS 集群上的 EC2 实例级指标
- 设置 Firelens 以向 CloudWatch Logs 发送日志

在 Amazon ECS 上针对集群级别和服务级别指标设置 Container Insights

可以在新的和现有的 Amazon ECS 集群上启用 Container Insights。Container Insights 在集群、任务和服务级别收集指标。您可以使用 Amazon ECS 控制台或 AWS CLI 启用 Container Insights。

如果您在 Amazon EC2 实例上使用 Amazon ECS，并且您要收集 Container Insights 中的网络和存储指标，则必须使用包含 Amazon ECS 代理版本 1.29 的 AMI 启动该实例。有关更新代理版本的信息，请参阅更新 Amazon ECS 容器代理

您可以使用 AWS CLI 设置账户级别权限，以便为您账户中创建的任何新 Amazon ECS 集群启用 Container Insights。为此，请输入以下命令。

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

> **Note**
>
> 如果您用于 Amazon ECS Container Insights 指标的客户自主管理型 AWS KMS 密钥尚未配置为与 CloudWatch 配合使用，则必须更新密钥政策以允许在 CloudWatch Logs 中使用加密日志。您还必须将自己的 AWS KMS 密钥与 `/aws/ecs/containerinsights/ClusterName/performance` 下的日志组相关联。有关更多信息，请参阅使用 AWS Key Management Service 对 CloudWatch Logs 中的日志数据进行加密。

在现有 Amazon ECS 集群上设置 Container Insights

要在现有 Amazon ECS 集群上启用 Container Insights，请输入以下命令。您必须运行版本 1.16.200 或更高版本的 AWS CLI，才能正常运行以下命令。

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=enabled
```

在新的 Amazon ECS 集群上设置 Container Insights

可通过两种方式在新的 Amazon ECS 集群上启用 Container Insights。您可以配置 Amazon ECS，以便默认情况下为所有新集群启用 Container Insights。否则，您可以在创建新集群时启用它。

使用 AWS Management Console

默认情况下，您可以在所有新集群上启用 Container Insights，也可以在创建单个集群时对其启用 Container Insights。

在默认情况下在所有新集群上启用 Container Insights

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
2. 在导航页面中，选择 Account Settings（账户设置）。
3. 选择更新。
4. 要默认使用集群的 CloudWatch Container Insights，请在 CloudWatch Container Insights 下选择或清除 CloudWatch Container Insights。
5. 选择 Save changes（保存更改）。

如果您在默认情况下未使用上述过程在所有新集群上启用 Container Insights，则使用以下步骤创建一个启用了 Container Insights 的集群。

创建启用了 Container Insights 的集群

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
2. 在导航窗格中，选择集群。
3. 在 Clusters（集群）页面上，选择 Create cluster（创建集群）。
4. 在 Cluster configuration（集群配置）下，为 Cluster name（集群名称），输入唯一名称。

   该名称最多可以包含 255 个字母（大小写字母）、数字和连字符。
5. 要打开 Container Insights，请展开监控，然后打开使用 Container Insights。

现在，您可以在集群中创建任务定义、运行任务和启动服务。有关更多信息，请参阅下列内容：

- 创建任务定义
- 正在运行的任务
- 创建服务

使用 AWS CLI 在新的 Amazon ECS 集群上设置 Container Insights

要在默认情况下在所有新集群上启用 Container Insights，请输入以下命令。

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

如果您在默认情况下未使用上述命令在所有新集群上启用 Container Insights，请输入以下命令来创建一个启用了 Container Insights 的新集群。您必须运行版本 1.16.200 或更高版本的 AWS CLI，才能正常运行以下命令。

```
aws ecs create-cluster --cluster-name myCICluster --settings
 "name=containerInsights,value=enabled"
```

在 Amazon ECS 集群上禁用 Container Insights

要在现有 Amazon ECS 集群上禁用 Container Insights，请输入以下命令。

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=disabled
```

使用 AWS Distro for OpenTelemetry 在 Amazon ECS 上设置 Container Insights

如果需要使用 AWS Distro for OpenTelemetry 在 Amazon ECS 集群上设置 CloudWatch Container Insights，请使用此部分。有关 AWS Distro for Open Telemetry 的更多信息，请参阅 AWS Distro for OpenTelemetry。

这些步骤假设您已具有正在运行 Amazon ECS 的集群。有关将 AWS Distro for Open Telemetry 与 Amazon ECS 结合使用以及为此设置 Amazon ECS 集群的更多信息，请参阅在 Amazon Elastic Container Service 中设置 AWS Distro for OpenTelemetry Collector。

步骤 1：创建任务角色

第一步是在集群中创建 AWS OpenTelemetry Collector 将使用的任务角色。

为 AWS Distro for OpenTelemetry 创建任务角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中选择策略，然后选择创建策略。
3. 选择 JSON 选项卡，然后复制以下策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

4. 选择查看策略。
5. 对于名称，输入 **AWSDistroOpenTelemetryPolicy**，然后选择 Create policy（创建策略）。
6. 在左侧的导航窗格中，选择 Roles（角色），然后选择 Create role（创建角色）。
7. 在服务列表中，选择 Elastic Container Service（Elastic Container 服务）。
8. 在页面下方，选择 Elastic Container Service Task（Elastic Container 服务任务），然后选择 Next: Permissions（下一步：权限）。
9. 在策略列表中，搜索 AWSDistroOpenTelemetryPolicy。
10. 选中 AWSDistroOpenTelemetryPolicy 旁的复选框。
11. 依次选择 Next: Tags（下一步：标签）和 Next: Review（下一步：查看）。
12. 对于 Role name（角色名称），输入 **AWSOpenTelemetryTaskRole**，然后选择 Create role（创建角色）。

步骤 2：创建任务执行角色

下一步是为 AWS OpenTelemetry Collector 创建任务执行角色。

为 AWS Distro for OpenTelemetry 创建任务执行角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧的导航窗格中，选择 Roles（角色），然后选择 Create role（创建角色）。
3. 在服务列表中，选择 Elastic Container Service（Elastic Container 服务）。
4. 在页面下方，选择 Elastic Container Service Task（Elastic Container 服务任务），然后选择 Next: Permissions（下一步：权限）。

5. 在策略列表中，搜索 AmazonECSTaskExecutionRolePolicy，然后选中 AmazonECSTaskExecutionRolePolicy 旁的复选框。
6. 在策略列表中，搜索 CloudWatchLogsFullAccess，然后选中 CloudWatchLogsFullAccess 旁的复选框。
7. 在策略列表中，搜索 AmazonSSMReadOnlyAccess，然后选中 AmazonSSMReadOnlyAccess 旁的复选框。
8. 依次选择 Next: Tags（下一步：标签）和 Next: Review（下一步：查看）。
9. 对于 Role name（角色名称），输入 **`AWSOpenTelemetryTaskExecutionRole`**，然后选择 Create role（创建角色）。

步骤 3：创建任务定义

下一步是创建任务定义。

为 AWS Distro for OpenTelemetry 创建任务定义

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
2. 在导航窗格中，选择 Task definitions（任务定义）
3. 选择 Create new task definition（创建新的任务定义）、Create new task definition（创建新的任务定义）。
4. 对于Task definition family（任务定义系列）中，为任务定义指定唯一名称。
5. 配置您的容器，然后选择下一步。
6. 在指标和日志记录下，选择使用指标收集。
7. 选择下一步。
8. 选择创建。

有关将 AWS OpenTelemetry 收集器与 Amazon ECS 结合使用的更多信息，请参阅在 Amazon Elastic Container Service 中设置 AWS Distro for OpenTelemetry Collector。

步骤 4：运行任务

最后一步是运行您创建的任务。

运行 AWS Distro for OpenTelemetry 的任务

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。

2. 在左侧导航窗格中，选择 Task Definitions（任务定义），然后选择您刚刚创建的任务。
3. 选择操作、部署、运行任务。
4. 选择 Deploy（部署）、Run task（运行任务）。
5. 在计算选项部分中，从现有集群中选择集群。
6. 选择创建。
7. 接下来，您可以在 CloudWatch 控制台中检查新指标。
8. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
9. 在左侧导航窗格中，选择 Metrics。

   您会看到 ECS/ContainerInsights 命名空间。选择该命名空间，然后您会看到八个指标。

部署 CloudWatch 代理以收集 Amazon ECS 上的 EC2 实例级别指标

要部署 CloudWatch 代理以从 EC2 实例上托管的 Amazon ECS 集群中收集实例级别指标，请使用具有默认配置的快速启动设置，或手动安装该代理以便对其进行自定义。

这两种方法都要求您已经部署了至少一个带有 EC2 启动类型的 Amazon ECS 集群，并且 CloudWatch 代理容器可以访问 Amazon EC2 实例元数据服务（IMDS）。有关 IMDS 的更多信息，请参阅实例元数据和用户数据。

这些方法还假设您已安装 AWS CLI。此外，要在以下过程中运行这些命令，您必须登录到具有 IAMFullAccess 和 AmazonECS_FullAccess 策略的账户或角色。

主题

- 使用 AWS CloudFormation 进行快速设置
- 手动和自定义设置

使用 AWS CloudFormation 进行快速设置

要使用快速设置，请输入以下命令来使用 AWS CloudFormation 安装代理。将 *cluster-name* 和 *cluster-region* 分别替换为您的 Amazon ECS 集群的名称和区域。

此命令将创建 IAM 角色，即 CWAgentECSTaskRole 和 CWAgentECSExecutionRole。如果您的账户中已有这些角色，请在输入命令时使用 `ParameterKey=CreateIAMRoles,ParameterValue=False` 而非 `ParameterKey=CreateIAMRoles,ParameterValue=True`。否则，此命令将失败。

```
ClusterName=cluster-name
Region=cluster-region
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

（备选方案）使用您自己的 IAM 角色

如果您要使用自己的自定义 ECS 任务角色和 ECS 任务执行角色，而不是使用 CWAgentECSTaskRole 和 CWAgentECSExecutionRole 角色，请先确保要用作 ECS 任务角色的角色已附加 CloudWatchAgentServerPolicy。此外，请确保要用作 ECS 任务执行角色的角色同时附加了 CloudWatchAgentServerPolicy 和 AmazonECSTaskExecutionRolePolicy 策略。然后，输入以下命令。在该命令中，将 *task-role-arn* 替换为您的自定义 ECS 任务角色的 ARN，并将 *execution-role-arn* 替换为您的自定义 ECS 任务执行角色的 ARN。

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
    --template-body file://cwagent-ecs-instance-metric-cfn.json \
    --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
                 ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${Region}
```

对快速设置进行故障排除

要查看 AWS CloudFormation 堆栈的状态，请输入以下命令。

```
ClusterName=cluster-name
```

```
Region=cluster-region
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --
region $Region
```

如果您看到的 StackStatus 不是 CREATE_COMPLETE 或 CREATE_IN_PROGRESS，请查看堆栈事件以查找错误。输入以下 命令。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region
 --region $Region
```

要查看 cwagent 守护进程服务的状态，请输入以下命令。在输出中，您应看到 deployment 部分中的 runningCount 等于 desiredCount。如果二者不相等，请查看输出中的 failures 部分。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --
region $Region
```

您还可以使用 CloudWatch Logs 控制台来查看代理日志。查找 /ecs/ecs-cwagent-daemon-service 日志组。

删除 CloudWatch 代理的 AWS CloudFormation 堆栈

如果您需要删除 AWS CloudFormation 堆栈，请输入以下命令。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --
region ${Region}
```

手动和自定义设置

执行此部分中的步骤，手动部署 CloudWatch 代理以从 EC2 实例上托管的 Amazon ECS 集群收集实例级别指标。

必要的 IAM 角色和策略

需要两个 IAM 角色。如果这两个角色不存在，则必须创建它们。有关这些角色的更多信息，请参阅任务的 IAM 角色和 Amazon ECS 任务执行角色。

- 一个 ECS 任务角色，可供 CloudWatch 代理用来发布指标。如果此角色已存在，则必须确保它附加了 CloudWatchAgentServerPolicy 策略。
- 一个 ECS 任务执行角色，可供 Amazon ECS 代理用来启动 CloudWatch 代理。如果此角色已存在，则必须确保它附加了 AmazonECSTaskExecutionRolePolicy 和 CloudWatchAgentServerPolicy 策略。

如果您还没有这些角色，则可使用以下命令来创建它们并附加必要的策略。第一条命令将创建 ECS 任务角色。

```
aws iam create-role --role-name CWAgentECSTaskRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

在输入上一条命令后，将命令输出中的 Arn 的值记为“TaskRoleArn”。稍后，您在创建任务定义时将需要使用它。然后，输入以下命令来附加必要的策略。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSTaskRole
```

下一条命令将创建 ECS 任务执行角色。

```
aws iam create-role --role-name CWAgentECSExecutionRole \
    --assume-role-policy-document "{\"Version\": \"2012-10-17\",\"Statement\": [{\"Sid
\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com
\"},\"Action\": \"sts:AssumeRole\"}]}"
```

在输入上一条命令后，将命令输出中的 Arn 的值记为“ExecutionRoleArn”。稍后，您在创建任务定义时将需要使用它。然后，输入以下命令来附加必要的策略。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
    --role-name CWAgentECSExecutionRole

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
    --role-name CWAgentECSExecutionRole
```

创建任务定义并启动守护进程服务

创建一个任务定义并使用它将 CloudWatch 代理作为守护进程服务启动。要创建任务定义，请输入以下命令。在前面的行中，将占位符替换为部署的实际值。*logs-region* 为 CloudWatch Logs 所在的区域，*cluster-region* 为集群所在的区域。*task-role-arn* 为您使用的 ECS 任务角色的 Arn，*execution-role-arn* 为 ECS 任务执行角色的 Arn。

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

然后，运行以下命令来启动守护程序服务。将 *cluster-name* 和 *cluster-region* 分别替换为您的 Amazon ECS 集群的名称和区域。

> **Important**
>
> 在运行此命令之前，请移除所有容量提供程序策略。否则，命令将无法正常运行。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

如果您看到的错误消息为 `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`，则您已创建名为 `cwagent-daemon-service` 的守护程序服务。您必须先删除该服务，并使用以下命令作为示例。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs delete-service \
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

（可选）高级配置

（可选）您可以使用 SSM 来指定 EC2 实例上托管的 Amazon ECS 集群中 CloudWatch 代理的其他配置选项。这些选项如下所示：

- `metrics_collection_interval` – CloudWatch 代理收集指标的频率（以秒为单位）。默认值为 60。范围为 1 – 172000。
- `endpoint_override` –（可选）指定要将日志发送到的其他端点。如果您从 VPC 的集群中发布并希望将日志数据传输到 VPC 终端节点，则可能需要执行该操作。

  `endpoint_override` 的值必须是表示 URL 的字符串。
- `force_flush_interval` – 以秒为单位指定日志在发送到服务器之前保留在内存缓冲区中的最大时间量。无论此字段的设置如何，如果缓冲区中的日志大小达到 1 MB，日志会立即发送到服务器。默认值为 5 秒。
- `region` – 默认情况下，代理将指标发布到 Amazon ECS 容器实例所在的同一区域。要覆盖此设置，您可以在此处指定其他区域。例如，`"region" : "us-east-1"`

以下是自定义配置的示例：

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "ecs": {
                "metrics_collection_interval": 30
            }
        },
        "force_flush_interval": 5
    }
```

```
}
```

在 Amazon ECS 容器中自定义 CloudWatch 代理配置

1. 确保将 AmazonSSMReadOnlyAccess 策略附加到 Amazon ECS 任务执行角色。可输入以下命令来执行此操作。此示例假定您的 Amazon ECS 任务执行角色是 CWAgentECSExecutionRole。如果您使用其他角色，请在以下命令中替换该角色名称。

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonSSMReadOnlyAccess \
           --role-name CWAgentECSExecutionRole
   ```

2. 创建与上一示例类似的自定义配置文件。将此文件命名为 `/tmp/ecs-cwagent-daemon-config.json`。

3. 运行以下命令以将此配置放入 Parameter Store 中。将 *`cluster-region`* 替换为 Amazon ECS 集群的区域。要运行此命令，您必须登录到具有 AmazonSSMFullAccess 策略的用户或角色。

   ```
   Region=cluster-region
   aws ssm put-parameter \
       --name "ecs-cwagent-daemon-service" \
       --type "String" \
       --value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
       --region $Region
   ```

4. 将任务定义文件下载到本地文件，例如 `/tmp/cwagent-ecs-instance-metric.json`

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
   insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/
   cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-
   instance-metric.json
   ```

5. 修改任务定义文件。删除以下部分：

   ```
   "environment": [
                   {
                       "name": "USE_DEFAULT_CONFIG",
                       "value": "True"
                   }
               ],
   ```

   将此部分替换为以下内容：

```
"secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "ecs-cwagent-daemon-service"
                }
            ],
```

6. 通过执行以下步骤，将代理作为守护程序服务重新启动：

   a. 运行以下命令。

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
    | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
    | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-
json
```

   b. 运行以下命令来启动守护程序服务。将 *cluster-name* 和 *cluster-region* 分别替换为您的 Amazon ECS 集群的名称和区域。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
    --cluster ${ClusterName} \
    --service-name cwagent-daemon-service \
    --task-definition ecs-cwagent-daemon-service \
    --scheduling-strategy DAEMON \
    --region ${Region}
```

   如果您看到的错误消息为 `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`，则您已创建名为 `cwagent-daemon-service` 的守护程序服务。您必须先删除该服务，并使用以下命令作为示例。

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
```

```
    --cluster ${ClusterName} \
    --service cwagent-daemon-service \
    --region ${Region} \
    --force
```

部署 AWS Distro for OpenTelemetry 以收集 Amazon ECS 集群上的 EC2 实例级指标

使用此部分中的步骤使用 AWS Distro for OpenTelemetry 来收集 Amazon ECS 集群上的 EC2 实例级指标。有关 AWS Distro for OpenTelemetry 的更多信息，请参阅 AWS Distro for OpenTelemetry。

这些步骤假设您已具有正在运行 Amazon ECS 的集群。此集群必须使用 EC2 启动类型进行部署。有关将 AWS Distro for Open Telemetry 与 Amazon ECS 结合使用以及为此设置 Amazon ECS 集群的更多信息，请参阅设置 Amazon Elastic Container Service 中的 AWS Distro for OpenTelemetry Collector 以了解 ECS EC2 实例级指标。

主题

- 使用 AWS CloudFormation 进行快速设置
- 手动和自定义设置

使用 AWS CloudFormation 进行快速设置

下载 AWS CloudFormation 模板文件，用于在 EC2 上为 Amazon ECS 安装 AWS Distro for OpenTelemetry 收集器。运行以下 curl 命令。

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

下载模板文件后，将其打开并将 *PATH_TO_CloudFormation_TEMPLATE* 替换为您保存模板文件的路径。然后导出以下参数并运行 AWS CloudFormation 命令，如以下命令所示。

- Cluster_Name– Amazon ECS 集群名称
- AWS_Region– 将发送数据的区域
- PATH_TO_CloudFormation_TEMPLATE– 保存 AWS CloudFormation 模板文件的路径。
- command– 要启用 AWS Distro for OpenTelemetry 收集器来收集 Amazon EC2 上 Amazon ECS 的实例级指标，您必须指定 `--config=/etc/ecs/otel-instance-metrics-config.yaml` 作为此参数。

```
ClusterName=Cluster_Name
Region=AWS_Region
command=--config=/etc/ecs/otel-instance-metrics-config.yaml
aws cloudformation create-stack --stack-name AOCECS-${ClusterName}-${Region} \
--template-body file://PATH_TO_CloudFormation_TEMPLATE \
--parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
ParameterKey=CreateIAMRoles,ParameterValue=True \
ParameterKey=command,ParameterValue=${command} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${Region}
```

运行此命令后，使用 Amazon ECS 控制台查看任务是否正在运行。

对快速设置进行故障排除

要查看 AWS CloudFormation 堆栈的状态，请输入以下命令。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region
 $Region
```

如果 StackStatus 的值不是 CREATE_COMPLETE 或 CREATE_IN_PROGRESS，请查看堆栈事件以查找错误。输入以下 命令。

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --
region $Region
```

要查看 AOCECS 守护进程服务的状态，请输入以下命令。在输出中，您应看到部署部分中的 desiredCount 等于 runningCount。如果二者不相等，请查看输出中的故障部分。

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --
region $Region
```

您还可以使用 CloudWatch Logs 控制台来查看代理日志。查找 /aws/ecs/containerinsights/{ClusterName}/performance 日志组。

手动和自定义设置

执行此部分中的步骤，手动部署 AWS Distro for OpenTelemetry 以从 Amazon EC2 实例上托管的 Amazon ECS 集群收集实例级别指标。

步骤 1：必要的角色和策略

需要两个 IAM 角色。如果这两个角色不存在，则必须创建它们。有关这些角色的更多信息，请参阅创建 IAM 策略和创建 IAM 角色。

步骤 2：创建任务定义

创建一个任务定义并使用它将 AWS Distro for OpenTelemetry 作为守护进程服务启动。

要使用任务定义模板创建任务定义，请按照使用 AWS OTel Collector 为 EC2 实例创建 ECS EC2 任务定义中的说明进行操作。

要使用 Amazon ECS 控制台创建任务定义，请按照通过 AWS 控制台为 Amazon ECS EC2 实例指标创建任务定义来安装 AWS OTel Collector 中的说明进行操作。

步骤 3：启动守护进程服务

要将 AWS Distro for OpenTelemetry 作为守护进程服务启动，请按照使用守护进程服务在 Amazon Elastic Container Service (Amazon ECS) 上运行您的任务中的说明进行操作。

（可选）高级配置

（可选）您可以使用 SSM 来指定 Amazon EC2 实例上托管的 Amazon ECS 集群中 AWS Distro for OpenTelemetry 的其他配置选项。有关创建配置文件的更多信息，请参阅自定义 OpenTelemetry 配置。有关在配置文件中可使用的选项的详细信息，请参阅 AWS Container Insights Receiver。

设置 Firelens 以向 CloudWatch Logs 发送日志

FireLens for Amazon ECS 使您能够使用任务定义参数将日志路由到 Amazon CloudWatch Logs，以进行日志存储和分析。FireLens 与 Fluent Bit 和 Fluentd 配合使用。我们提供 AWS for Fluent Bit 镜像，您也可以使用自己的 Fluent Bit 或 Fluentd 镜像。支持通过 AWS SDK、AWS CLI 和 AWS Management Console 使用 FireLens 配置创建 Amazon ECS 任务定义。有关 CloudWatch Logs 的更多信息，请参阅什么是 CloudWatch Logs？。

使用 FireLens for Amazon ECS 时有一些关键注意事项。有关更多信息，请参阅注意事项。

要查找 AWS for Fluent Bit 镜像，请参阅使用 AWS for Fluent Bit 镜像。

要创建使用 FireLens 配置的任务定义，请参阅创建使用 FireLens 配置的任务定义。

示例

以下任务定义示例演示如何指定用于将日志转发到 CloudWatch Logs 日志组的日志配置。如需了解更多信息，请参阅 Amazon CloudWatch Logs 用户指南中的什么是 Amazon CloudWatch Logs？

在日志配置选项中，指定日志组名称及其所在的区域。要让 Fluent Bit 代表您创建日志组，请指定 `"auto_create_group":"true"`。您还可以将任务 ID 指定为有助于筛选的日志流前缀。有关更多信息，请参阅适用于 CloudWatch Logs 的 Fluent Bit 插件。

```
{
 "family": "firelens-example-cloudwatch",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "nginx",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "cloudwatch_logs",
     "region": "us-west-2",
```

```
      "log_key": "log",
                                  "log_group_name": "/aws/ecs/containerinsights/
$(ecs_cluster)/application",
      "auto_create_group": "true",
      "log_stream_name": "$(ecs_task_id)"
     }
    },
    "memoryReservation": 100
   }
  ]
}
```

在 Amazon EKS 和 Kubernetes 上设置 Container Insights

Amazon EKS 版本 1.23 及更高版本支持 Container Insights。仅版本 1.24 及更高版本支持快速启动安装方法。

在 Amazon EKS 或 Kubernetes 上设置 Container Insights 的整个过程如下所示：

1. 验证您是否满足所需的先决条件。
2. 在集群上设置 Amazon CloudWatch Observability EKS 附加组件、CloudWatch 代理或 AWS Distro for OpenTelemetry，以将指标发送到 CloudWatch。

> **Note**
>
> 要使用针对 Amazon EKS 增强了可观测性的 Container Insights，您必须使用 Amazon CloudWatch Observability EKS 附加组件或 CloudWatch 代理。有关此版本的 Container Insights 的更多信息，请参阅 针对 Amazon EKS 增强了可观测性的 Container Insights。要将 Container Insights 与 Fargate 一起使用，您必须使用 AWS Distro for OpenTelemetry。Fargate 不支持针对 Amazon EKS 增强了可观测性的 Container Insights。

> **Note**
>
> Container Insights 现在支持 Amazon EKS 集群中的 Windows Worker 节点。Windows 还支持针对 Amazon EKS 增强了可观测性的 Container Insights。有关在 Windows 上启用 Container Insights 的信息，请参阅 使用启用了 Container Insights 增强可观测性的 CloudWatch 代理。

设置 Fluent Bit 或 Fluentd 以将日志发送到 CloudWatch Logs。（如果您安装了 Amazon CloudWatch Observabity EKS 附加组件，则默认启用此功能。）

如果您正在使用 CloudWatch 代理，您可以作为快速入门设置的一部分立即执行这些步骤，也可以单独执行这些步骤。

3. （可选）设置 Amazon EKS 控制面板日志记录。
4. （可选）在集群上将 CloudWatch 代理设置为 StatsD 端点以将 StatsD 指标发送到 CloudWatch。
5. （可选）启用 App Mesh Envoy 访问日志。

在 Container Insights 的原始版本中，收集的指标和摄取的日志将作为自定义指标收费。借助针对 Amazon EKS 增强了可观测性的 Container Insights，Container Insights 指标和日志按每次观测收费，而不是按存储的指标或摄取的日志收费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

主题

- 验证 先决条件
- 使用启用了 Container Insights 增强可观测性的 CloudWatch 代理
- 使用 AWS Distro for OpenTelemetry
- 将日志发送到 CloudWatch Logs
- 在 Amazon EKS 和 Kubernetes 上更新或删除 Container Insights

验证 先决条件

在 Amazon EKS 或 Kubernetes 上安装 Container Insights 之前，请验证以下内容。无论您是使用 CloudWatch 代理还是 AWS Distro for OpenTelemetry 在 Amazon EKS 集群上设置 Container Insights，这些先决条件都适用。

- 在其中一个支持适用于 Amazon EKS 和 Kubernetes 的 Container Insights 的区域中，您具有正常工作的 Amazon EKS 集群或 Kubernetes 集群并附加了节点。有关支持的区域列表，请参阅Container Insights。
- 您已安装并正在运行 `kubectl`。有关更多信息，请参阅 Amazon EKS 用户指南 中的安装 `kubectl`。
- 如果您使用在 AWS 上运行的 Kubernetes，而不是使用 Amazon EKS，则还需要满足以下先决条件：

- 确保您的 Kubernetes 集群已启用基于角色的访问控制 (RBAC)。有关更多信息，请参阅《Kubernetes 参考》中的 使用 RBAC 授权。
- 您的 kubelet 已启用 Webhook 授权模式。有关更多信息，请参阅《Kubernetes 参考》中的 Kubelet 身份验证/授权。

您还必须授予 IAM 权限，以使 Amazon EKS Worker 节点能够向 CloudWatch 发送指标和日志。有两种方式可执行此操作：

- 将策略附加到 Worker 节点的 IAM 角色。这同时适用于 Amazon EKS 集群和其他 Kubernetes 集群。
- 对集群的服务账户使用 IAM 角色，并将策略附加到此角色。这仅适用于 Amazon EKS 集群。

第一个选项授予 CloudWatch 对整个节点的权限，而对服务账户使用 IAM 角色仅授予 CloudWatch 对相应守护程序集 Pod 的访问权限。

将策略附加到 Worker 节点的 IAM 角色

按照以下步骤将策略附加到 Worker 节点的 IAM 角色。这同时适用于 Amazon EKS 外部的 Amazon EKS 集群和 Kubernetes 集群。

将所需的策略附加到 Worker 节点的 IAM 角色

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 选择其中的一个 Worker 节点实例，然后在描述中选择 IAM 角色。
3. 在 IAM 角色页面上，选择 Attach policies（附加策略）。
4. 在策略列表中，选中 CloudWatchAgentServerPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
5. 选择附加策略。

如果您在 Amazon EKS 外部运行 Kubernetes 集群，则可能无法将 IAM 角色附加到 Worker 节点。如果没有，您必须先将一个 IAM 角色附加到实例，然后按照前面步骤中所述添加策略。有关将角色附加到实例的更多信息，请参阅《Amazon EC2 用户指南》中的将 IAM 角色附加到实例。

如果您在 Amazon EKS 外部运行 Kubernetes 集群，并且希望在指标中收集 EBS 卷 ID，则必须向附加到该实例的 IAM 角色添加另一个策略。添加以下内容作为内联策略。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 身份权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

使用 IAM 服务账户角色

此方法仅适用于 Amazon EKS 集群。

使用 IAM 服务账户角色向 CloudWatch 授予权限

1. 为集群上的服务账户启用 IAM 角色（如果尚未执行此操作）。有关更多信息，请参阅为集群上的服务账户启用 IAM 角色。
2. 配置服务账户以使用 IAM 角色（如果尚未进行此配置）。有关更多信息，请参阅配置 Kubernetes 服务账户以代入 IAM 角色。

   创建角色时，除了您为角色创建的策略外，还将 CloudWatchAgentServerPolicy IAM 策略附加到角色。此外，应在 `amazon-cloudwatch` 命名空间中创建链接至此角色的关联 Kubernetes 服务账户，在接下来的步骤中，CloudWatch 和 Fluent Bit 进程守护程序集将在其中部署
3. 将 IAM 角色与集群中的服务账户相关联（如果尚未执行此操作）。有关更多信息，请参阅配置 Kubernetes 服务账户以代入 IAM 角色。

使用启用了 Container Insights 增强可观测性的 CloudWatch 代理

按照以下任一部分中的说明，使用 CloudWatch 代理在 Amazon EKS 集群或 Kubernetes 集群上设置 Container Insights。仅 Amazon EKS 版本 1.24 及更高版本支持快速启动指令。

> **Note**
>
> 您可以按照以下任一部分中的说明安装 Container Insights。您无需遵循所有三组说明。

主题

- 安装 Amazon CloudWatch Observability EKS 附加组件
- Amazon EKS 和 Kubernetes 上的 Container Insights 的快速入门设置
- 设置 CloudWatch 代理以收集集群指标

安装 Amazon CloudWatch Observability EKS 附加组件

您可以使用 Amazon EKS 附加组件安装针对 Amazon EKS 增强了可观测性的 Container Insights。该附加组件会安装 CloudWatch 代理以从集群发送基础设施指标、安装 Fluent Bit 以发送容器日志，还会启用 CloudWatch Application Signals 以发送应用程序性能遥测。

当您使用 Amazon EKS 附加组件 1.5.0 版或更高版本时，集群中的 Linux 和 Windows Worker 节点上都将启用 Container Insights。目前，Amazon EKS 中的 Windows 上不支持 Application Signals。

运行 Kubernetes 而不是 Amazon EKS 的集群不支持 Amazon EKS 附加组件。

有关 Amazon CloudWatch Observability EKS 附加组件的更多信息，请参阅 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理。

安装 Amazon CloudWatch Observability EKS 附加组件

1. 首先，通过将 CloudWatchAgentServerPolicy IAM 策略附加到您的 Worker 节点来设置所需权限。为此，请输入以下命令。将 *my-worker-node-role* 替换为您的 Kubernetes Worker 节点使用的 IAM 角色。

   ```
   aws iam attach-role-policy \
   --role-name my-worker-node-role \
   --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
   ```

2. 输入以下命令以安装附加组件：

   ```
   aws eks create-addon --cluster-name my-cluster-name --addon-name amazon-cloudwatch-observability
   ```

Amazon EKS 和 Kubernetes 上的 Container Insights 的快速入门设置

> **Important**
>
> 如果您要在 Amazon EKS 集群上安装 Container Insights，我们建议您使用 Amazon CloudWatch Observability EKS 附加组件进行安装，而不是按照本部分中的说明进行安装。此外，要检索加速计算网络，必须使用 Amazon CloudWatch 可观测性 EKS 附加组件。有关更多信息和说明，请参阅 安装 Amazon CloudWatch Observability EKS 附加组件。

要完成 Container Insights 的设置，您可以按照本节中的快速入门说明进行操作。如果您要在 Amazon EKS 集群中进行安装，并且在 2023 年 11 月 6 日当天或之后按照本部分中的说明进行操作，则可以在集群中安装针对 Amazon EKS 增强了可观测性的 Container Insights。

> **Important**
>
> 在完成本节中的步骤之前，您必须已对 IAM 权限等先决条件进行验证。有关更多信息，请参阅 验证 先决条件。

或者，您可以按照以下两节中的说明操作：设置 CloudWatch 代理以收集集群指标和 将日志发送到 CloudWatch Logs。这些章节提供有关 CloudWatch 代理如何与 Amazon EKS 和 Kubernetes 结合使用的更多配置详细信息，但需要您执行更多安装步骤。

在 Container Insights 的原始版本中，收集的指标和摄取的日志将作为自定义指标收费。借助针对 Amazon EKS 增强了可观测性的 Container Insights，Container Insights 指标和日志按每次观测收费，而不是按存储的指标或摄取的日志收费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> Amazon 现已推出 Fluent Bit 作为 Container Insights 的默认日志解决方案，其性能显着提高。我们建议您使用 Fluent Bit 而不是 Fluentd。

使用 CloudWatch 代理 operator 和 Fluent Bit 快速入门

Fluent Bit 有两种配置：优化版本和更类似于 Fluentd 的体验的版本。快速入门配置使用优化版本。有关 Fluentd 兼容配置的详细信息，请参阅 将 Fluent Bit 设置为 DaemonSet 以将日志发送到 CloudWatch Logs。

CloudWatch 代理 operator 是安装到 Amazon EKS 集群的附加容器。它以 OpenTelemetry Operator for Kubernetes 为模型。Operator 管理集群中 Kubernetes 资源的生命周期。它在 Amazon EKS 集群上安装 CloudWatch 代理、DCGM Exporter（NVIDIA）和 AWS Neuron Monitor 并对其进行管理。Fluent Bit 和 CloudWatch Agent for Windows 可以直接安装到 Amazon EKS 集群中，而无需 operator 对其进行管理。

为了获得更安全、功能更丰富的凭证颁发机构解决方案，CloudWatch 代理 operator 需要使用 cert-manager，这是 Kubernetes 中广泛采用的 TLS 证书管理解决方案。使用 cert-manager 可简化获取、续订、管理和使用这些证书的过程。此解决方案可确保证书有效且最新，并尝试在证书到期前的配置时间续订证书。cert-manager 还便于从各种支持的来源（包括 AWS Certificate Manager Private Certificate Authority）颁发证书。

使用快速入门部署 Container Insights

1. 如果 cert-manager 尚未安装到集群中，则请安装。有关更多信息，请参阅 cert-manager Installation。
2. 通过输入以下命令安装自定义资源定义（CRD）。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl
 apply --server-side -f -
```

3. 通过输入以下命令安装 operator。将 *my-cluster-name* 替换为 Amazon EKS 或 Kubernetes 集群的名称，将 *my-cluster-region* 替换为在其中发布日志的区域的名称。我们建议您使用在其中部署集群的同一个区域来降低 AWS 出站数据传输成本。

```
ClusterName=my-cluster-name
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

例如，要在名为 MyCluster 的集群上部署 Container Insights 并将日志和指标发布到美国西部（俄勒冈），请输入以下命令。

```
ClusterName='MyCluster'
RegionName='us-west-2'
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 apply -f -
```

从 Container Insights 迁移

如果您已经在 Amazon EKS 集群中配置了 Container Insights，并想要迁移到针对 Amazon EKS 增强了可观测性的 Container Insights，则请参阅 升级到针对 Amazon EKS 增强了可观测性的 Container Insights

删除 Container Insights

如果您想使用快速入门设置删除 Container Insights，则请输入以下命令。

```
ClusterName=my-cluster-name
RegionName=my-cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
 delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete
 -f -
```

设置 CloudWatch 代理以收集集群指标

> **Important**
>
> 如果您要在 Amazon EKS 集群上安装 Container Insights，我们建议您使用 Amazon CloudWatch Observability EKS 附加组件进行安装，而不是按照本部分中的说明进行安装。有关更多信息和说明，请参阅 安装 Amazon CloudWatch Observability EKS 附加组件。

要设置 Container Insights 以收集指标，您可以按照Amazon EKS 和 Kubernetes 上的 Container Insights 的快速入门设置中的步骤操作，也可以按照本节中的步骤操作。在以下步骤中，您设置 CloudWatch 代理以便能够从集群中收集指标。

如果您要在 Amazon EKS 集群中进行安装，并且在 2023 年 11 月 6 日当天或之后按照本部分中的说明进行操作，则可以在集群中安装针对 Amazon EKS 增强了可观测性的 Container Insights。

步骤 1：为 CloudWatch 创建命名空间

使用以下步骤为 CloudWatch 创建名为 `amazon-cloudwatch` 的 Kubernetes 命名空间。如果已创建该命名空间，您可以跳过此步骤。

为 CloudWatch 创建命名空间

- 输入以下 命令。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

步骤 2：在集群中创建服务账户

如果您还没有服务账户，请使用以下方法之一为 CloudWatch 代理创建一个服务账户。

- 使用 `kubectl`
- 使用 `kubeconfig` 文件。

使用 **`kubectl`** 进行身份验证

要使用 **`kubectl`** 为 CloudWatch 代理创建服务账户

- 输入以下 命令。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-serviceaccount.yaml
```

如果您没有按照前面的步骤进行操作，而是已具有要使用的 CloudWatch 代理的服务账户，则必须确保该账户具有以下规则。此外，在 Container Insights 安装的其余步骤中，您必须使用该服务账户的名称，而不是 `cloudwatch-agent`。

```
rules:
  - apiGroups: [""]
    resources: ["pods", "nodes", "endpoints"]
    verbs: ["list", "watch"]
  - apiGroups: [ "" ]
    resources: [ "services" ]
    verbs: [ "list", "watch" ]
  - apiGroups: ["apps"]
    resources: ["replicasets", "daemonsets", "deployments", "statefulsets"]
    verbs: ["list", "watch"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["list", "watch"]
  - apiGroups: [""]
    resources: ["nodes/proxy"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes/stats", "configmaps", "events"]
    verbs: ["create", "get"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["cwagent-clusterleader"]
    verbs: ["get","update"]
  - nonResourceURLs: ["/metrics"]
    verbs: ["get", "list", "watch"]
```

使用 **kubeconfig** 进行身份验证

或者，您也可以使用 kubeconfig 文件进行身份验证。此方法让您可以直接在 CloudWatch 代理配置中指定 kubeconfig 路径，从而避免需要使用服务账户。它还允许您消除对 Kubernetes 控制面板 API 进行身份验证的依赖，从而简化设置，并通过以 kubeconfig 文件管理身份验证来潜在提高安全性。

要使用此方法，请更新您的 CloudWatch 代理配置文件以指定 kubeconfig 文件路径，如以下示例所示。

```
{
  "logs": {
```

```
    "metrics_collected": {
      "kubernetes": {
        "cluster_name": "YOUR_CLUSTER_NAME",
        "enhanced_container_insights": false,
        "accelerated_compute_metrics": false,
        "tag_service": false,
        "kube_config_path": "/path/to/your/kubeconfig"
        "host_ip": "HOSTIP"
      }
    }
  }
}
```

要创建 kubeconfig 文件，请为具有 system:masters Kubernetes 角色的 admin/{create_your_own_user} 用户创建证书签名请求（CSR）。然后，使用 Kubernetes 集群的证书颁发机构（CA）签名并创建 kubeconfig 文件。

步骤 3：为 CloudWatch 代理创建 ConfigMap

可以使用以下步骤为 CloudWatch 代理创建 ConfigMap。

为 CloudWatch 代理创建 ConfigMap

1. 运行以下命令以将 ConfigMap YAML 下载到 kubectl 客户端主机中：

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-configmap-enhanced.yaml
   ```

2. 编辑下载的 YAML 文件，如下所示：

   - cluster_name – 在 kubernetes 部分中，将 {{cluster_name}} 替换为您的集群的名称。删除 {{}} 字符。或者，如果使用的是 Amazon EKS 集群，您可以删除 "cluster_name" 字段和值。如果这样做，CloudWatch 代理将从 Amazon EC2 标签中检测集群名称。

3. （可选）根据您的监控要求，对 ConfigMap 进行进一步的更改，如下所示：

   - metrics_collection_interval – 在 kubernetes 部分中，您可以指定代理收集指标的频率。默认值为 60 秒。kubelet 中的默认 cadvisor 收集间隔为 15 秒，因此，请不要将该值设置为小于 15 秒。

- endpoint_override – 在 `logs` 部分中，如果要覆盖默认端点，您可以指定 CloudWatch Logs 端点。如果您从 VPC 的集群中发布并希望将数据传输到 VPC 终端节点，则可能需要执行该操作。
- force_flush_interval – 在 `logs` 部分中，您可以指定在将日志事件发布到 CloudWatch Logs 之前批量处理这些事件的间隔。默认值为 5 秒。
- region – 默认情况下，代理将指标发布到 Worker 节点所在的区域。要覆盖该区域，您可以在 `agent` 部分中添加 `region` 字段：例如，`"region":"us-west-2"`。
- statsd 部分 – 如果希望 CloudWatch Logs 代理还在集群的每个 Worker 节点中作为 StatsD 侦听器运行，您可以将 `statsd` 部分添加到 `metrics` 部分中，如以下示例中所示。有关该部分的其他 StatsD 选项的信息，请参阅 使用 StatsD 检索自定义指标 。

```
"metrics": {
  "metrics_collected": {
    "statsd": {
      "service_address":":8125"
    }
  }
}
```

JSON 部分的完整示例如下所示。如果您使用 kubeconfig 文件进行身份验证，请添加 kube_config_path 参数以指定 kubeconfig 文件的路径。

```
{
    "agent": {
        "region": "us-east-1"
    },
    "logs": {
        "metrics_collected": {
            "kubernetes": {
                "cluster_name": "MyCluster",
                "metrics_collection_interval": 60,
                "kube_config_path": "/path/to/your/kubeconfig" //if using
 kubeconfig for authentication
            }
        },
        "force_flush_interval": 5,
        "endpoint_override": "logs.us-east-1.amazonaws.com"
    },
    "metrics": {
        "metrics_collected": {
```

```
            "statsd": {
                "service_address": ":8125"
            }
        }
    }
}
```

4. 运行以下命令以在集群中创建 ConfigMap。

```
kubectl apply -f cwagent-configmap.yaml
```

步骤 4：将 CloudWatch 代理部署为 DaemonSet

要完成 CloudWatch 代理安装并开始收集容器指标，请使用以下步骤。

将 CloudWatch 代理部署为 DaemonSet

1. • 如果您不想在集群上使用 StatsD，请输入以下命令。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-
daemonset.yaml
```

   • 如果您想要使用 StatsD，请执行以下步骤：

      a. 运行以下命令以将 DaemonSet YAML 下载到 kubectl 客户端主机中。

```
curl -O  https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-
mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

      b. 取消注释 port 文件中的 cwagent-daemonset.yaml 部分，如下所示：

```
ports:
  - containerPort: 8125
    hostPort: 8125
    protocol: UDP
```

      c. 运行以下命令以在集群中部署 CloudWatch 代理。

```
kubectl apply -f cwagent-daemonset.yaml
```

d. 运行以下命令以在集群中的 Windows 节点上部署 CloudWatch 代理。Windows 上的 CloudWatch 代理不支持 StatSD 侦听器。

```
kubectl apply -f cwagent-daemonset-windows.yaml
```

2. 运行以下命令以验证是否部署了该代理。

```
kubectl get pods -n amazon-cloudwatch
```

在完成后，CloudWatch 代理创建一个名为 `/aws/containerinsights/Cluster_Name/performance` 的日志组，并将性能日志事件发送到该日志组。如果还将该代理设置为 StatsD 侦听器，该代理还会在端口 8125（具有计划运行应用程序 pod 的节点的 IP 地址）上侦听 StatsD 指标。

故障排除

如果未正确部署该代理，请尝试执行以下操作：

- 运行以下命令以获取 pod 列表。

```
kubectl get pods -n amazon-cloudwatch
```

- 运行以下命令并在输出底部检查事件。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- 运行以下命令以检查日志。

```
kubectl logs pod-name  -n amazon-cloudwatch
```

使用 AWS Distro for OpenTelemetry

您可以通过使用 AWS Distro for OpenTelemetry 收集器设置 Container Insights 从 Amazon EKS 集群收集指标。有关 AWS Distro for OpenTelemetry 的更多信息，请参阅 AWS Distro for OpenTelemetry。

> **Important**
>
> 如果您使用 AWS Distro for OpenTelemetry 进行安装，则会安装 Container Insights，但无法获取针对 Amazon EKS 增强了可观测性的 Container Insights。您将不会收集针对 Amazon EKS 增强了可观测性的 Container Insights 中支持的详细指标。

如何设置 Container Insights 取决于集群托管在 Amazon EC2 实例上还是 AWS Fargate (Fargate) 上。

托管在 Amazon EC2 上的 Amazon EKS 集群

如果您尚未执行此操作，请确保您已满足先决条件，例如必要的 IAM 角色。有关更多信息，请参阅 验证 先决条件。

Amazon 提供 Helm Chart，您可以用它来设置对 Amazon EC2 上的 Amazon Elastic Kubernetes Service 的监控。此监控使用 AWS Distro for OpenTelemetry(ADOT) 收集器用于指标，使用 Fluent Bit 用于日志。因此，Helm Chart 对使用 Amazon EC2 上的 Amazon EKS 并希望收集指标和日志以发送到 CloudWatch Container Insights 的客户非常有用。有关此 Helm Chart 的更多信息，请参阅发送到 Amazon CloudWatch Container Insights 的 EC2 上 EKS 的指标和日志的 ADOT Helm chart。

或者，您也可以使用本部分剩余内容中的说明。

首先，通过输入以下命令将 AWS Distro for OpenTelemetry 收集器部署为 DaemonSet。

```
curl https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/eks/otel-container-insights-infra.yaml |
kubectl apply -f -
```

要确认该收集器正在运行，请输入以下命令。

```
kubectl get pods -l name=aws-otel-eks-ci -n aws-otel-eks
```

如果此命令的输出包含处于 Running 状态的多个 Pod，则收集器正在运行并从集群收集指标。收集器会创建一个名为 aws/containerinsights/*cluster-name*/performance 的日志组并将性能日志事件发送给它。

有关如何在 CloudWatch 中查看 Container Insights 指标的信息，请参阅 查看 Container Insights 指标。

AWS 还在 GitHub 上为此情况提供了文档。如果要自定义 Container Insights 发布的指标和日志，请参阅 https://aws-otel.github.io/docs/getting-started/container-insights/eks-infra。

托管在 Fargate 上的 Amazon EKS 集群

有关如何配置和部署 ADOT 收集器以从部署到 Fargate 上的 Amazon EKS 集群的工作负载收集系统指标并将其发送到 CloudWatch Container Insights 的说明，请参阅 AWS Distro for OpenTelemetry 文档中的 Container Insights EKS Fargate。

将日志发送到 CloudWatch Logs

要将日志从您的容器发送到 Amazon CloudWatch Logs，您可以使用 Fluent Bit 或 Fluentd。有关更多信息，请参阅 Fluent Bit 和 Fluentd。

如果您尚未使用 Fluentd，出于以下原因，我们建议您使用 Fluent Bit：

- 与 Fluentd 相比，Fluent Bit 具有更小的资源占用空间，并且在内存和 CPU 使用率方面具有更高的资源效率。有关更详细比较，请参阅 Fluent Bit 与 Fluentd 性能比较。
- Fluent Bit 镜像由 AWS 开发和维护。这使得 AWS 能够采用新的 Fluent Bit 镜像功能并更快地响应问题。

主题

- Fluent Bit 与 Fluentd 性能比较
- 将 Fluent Bit 设置为 DaemonSet 以将日志发送到 CloudWatch Logs
- （可选）将 Fluentd 设置为 DaemonSet 以将日志发送到 CloudWatch Logs
- （可选）设置 Amazon EKS 控制面板日志记录
- （可选）启用 App Mesh Envoy 访问日志
- （可选）为大型集群启用 Use_Kubelet 功能

Fluent Bit 与 Fluentd 性能比较

下表显示了 Fluent Bit 在内存和 CPU 使用率方面优于 Fluentd 的性能优势。以下数字仅供参考，可能会因环境而异。

每秒日志数	Fluentd CPU 使用率	Fluent Bit CPU 使用率与 Fluentd 兼容配置	具有优化配置的 Fluent Bit CPU 使用率
100	0.35 vCPU	0.02 vCPU	0.02 vCPU

每秒日志数	Fluentd CPU 使用率	Fluent Bit CPU 使用率与 Fluentd 兼容配置	具有优化配置的 Fluent Bit CPU 使用率
1000	0.32 vCPU	0.14 vCPU	0.11 vCPU
5000	0.85 vCPU	0.48 vCPU	0.30 vCPU
10000	0.94 vCPU	0.60 vCPU	0.39 vCPU

每秒日志数	Fluentd 内存使用量	Fluent Bit 内存使用量与 Fluentd 兼容配置	优化配置的 Fluent Bit 内存使用量
100	153MB	46MB	37MB
1000	270MB	45MB	40MB
5000	320MB	55MB	45MB
10000	375MB	92MB	75MB

将 Fluent Bit 设置为 DaemonSet 以将日志发送到 CloudWatch Logs

以下部分帮助您部署 Fluent Bit 以将日志从容器发送到 CloudWatch Logs。

主题

- 您在使用 Fluentd 时出现的差异
- 设置 Fluent Bit
- 多行日志支持
- （可选）从 Fluent Bit 减少日志卷
- 故障排除
- 控制面板

您在使用 Fluentd 时出现的差异

如果您已经使用 Fluentd 将日志从容器发送到 CloudWatch Logs，请阅读本节以了解 Fluentd 和 Fluent Bit 之间的区别。如果您尚未将 Fluentd 与 Container Insights 结合使用，则可以跳至 设置 Fluent Bit。

我们为 Fluent Bit 提供了两种默认配置：

- Fluent Bit 优化配置 – 符合 Fluent Bit 最佳实践的配置。
- Fluentd 兼容配置 – 尽可能与 Fluentd 行为保持一致的配置。

下面的列表详细说明了 Fluentd 和每个 Fluent Bit 配置之间的差异。

- 日志流名称的差异 – 如果您使用 Fluent Bit 优化配置，日志流名称将有所不同。

  在 `/aws/containerinsights/Cluster_Name/application` 下

  - Fluent Bit 优化配置将日志发送到 *kubernetes-nodeName*-application.var.log.containers.*kubernetes-podName*_*kubernetes-namespace*_*kubernetes-container-name*-*kubernetes-containerID*
  - Fluentd 将日志发送到 *kubernetes-podName*_*kubernetes-namespace*_*kubernetes-containerName*_*kubernetes-containerID*

  在 `/aws/containerinsights/Cluster_Name/host` 下

  - Fluent Bit 优化配置将日志发送到 *kubernetes-nodeName*.*host-log-file*
  - Fluentd 将日志发送到 *host-log-file*-*Kubernetes-NodePrivateIp*

  在 `/aws/containerinsights/Cluster_Name/dataplane` 下

  - Fluent Bit 优化配置将日志发送到 *kubernetes-nodeName*.*dataplaneServiceLog*
  - Fluentd 将日志发送到 *dataplaneServiceLog*-*Kubernetes-nodeName*
- Container Insights 写入的 `kube-proxy` 和 `aws-node` 日志文件位于不同的位置。在 Fluentd 配置中，它们位于 `/aws/containerinsights/Cluster_Name/application`。在 Fluent Bit 优化配置中，它们位于 `/aws/containerinsights/Cluster_Name/dataplane`。
- Fluent Bit 和 Fluentd 中大部分元数据如 `pod_name` 和 `namespace_name` 是相同的，但以下是不同的。
  - Fluent Bit 优化配置使用 `docker_id`，而 Fluentd 使用 `Docker.container_id`。
  - 两种 Fluent Bit 配置均不使用以下元数据。它们仅存在于 Fluentd 中：`container_image_id`、`master_url`、`namespace_id` 和 `namespace_labels`。

设置 Fluent Bit

要设置 Fluent Bit 以从容器中收集日志，您可以按照 Amazon EKS 和 Kubernetes 上的 Container Insights 的快速入门设置 中的步骤操作，也可以按照本节中的步骤操作。

无论使用哪种方法，附加到集群节点的 IAM 角色必须具有足够的权限。有关运行 Amazon EKS 集群所需权限的更多信息，请参阅 Amazon EKS 用户指南中的 Amazon EKS IAM 策略、角色和权限。

在以下步骤中，您将 Fluent Bit 设置为 daemonSet 以将日志发送到 CloudWatch Logs。在完成该步骤时，Fluent Bit 将创建以下日志组（如果尚不存在）。

> **Important**
>
> 如果您已经在 Container Insights 中配置了 FluentD 并且 FluentD DaemonSet 没有按预期运行（如果您使用 containerd 运行时系统可能会发生这种情况），则必须在安装 Fluent Bit 之前将其卸载，以防止 Fluent Bit 处理 FluentD 错误日志消息。否则，您必须在成功安装 Fluent Bit 后立即卸载 FluentD。在安装 Fluent Bit 后卸载 Fluentd 可确保在此迁移过程中日志记录的连续性。将日志发送到 CloudWatch Logs 只需要 Fluent Bit 或 FluentD 中的其中一个。

日志组名称	日志源
/aws/containerinsights/*Cluster_Name* /application	/var/log/containers 中的所有日志文件
/aws/containerinsights/*Cluster_Name* /host	/var/log/dmesg 、/var/log/secure 和 /var/log/messages 中的日志
/aws/containerinsights/*Cluster_Name* /dataplane	kubelet.service 、kubeproxy .service 和 docker.service 的 /var/log/journal 中的日志。

安装 Fluent Bit 以将日志从容器发送到 CloudWatch Logs

1. 如果您还没有名为 amazon-cloudwatch 的命名空间，请通过输入以下命令创建一个：

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
   ```

2. 运行以下命令以创建一个名为 cluster-info 的 ConfigMap，该 ConfigMap 以集群名称和要向其发送日志的区域命名。将 *cluster-name* 和 *cluster-region* 分别替换为您的集群的名称和区域。

```
ClusterName=cluster-name
RegionName=cluster-region
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off'||
 FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||
 FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

在此命令中，默认情况下，用于监控插件指标的 `FluentBitHttpServer` 处于启用状态。要将其关闭，请将命令中的第三行更改为命令中的 `FluentBitHttpPort=''`（空字符串）。

同样，默认情况下，Fluent Bit 从尾部读取日志文件，并在部署后仅捕获新日志。如果你想要相反的设置，请设置 `FluentBitReadFromHead='On'`，它将收集文件系统中的所有日志。

3. 运行以下任一命令，将 Fluent Bit daemonset 下载并部署到集群中。

   - 如果需要针对 Linux 计算机的 Fluent Bit 优化配置，则请运行此命令。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-
     bit.yaml
     ```

   - 如果需要针对 Windows 计算机的 Fluent Bit 优化配置，则请运行此命令。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-
     windows.yaml
     ```

   - 如果使用 Linux 计算机且需要更类似于 Fluentd 的 Fluent Bit 配置，则请运行此命令。

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     ```

```
deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml
```

> **Important**
>
> 默认情况下，Fluent Bit 守护程序集配置会将日志级别设置为 INFO，这可能会导致 CloudWatch Logs 摄取成本增加。要减少日志摄取量和成本，您可以将日志级别更改为 ERROR。
> 有关如何减少日志量的更多信息，请参阅（可选）从 Fluent Bit 减少日志卷。

4. 运行以下命令以验证部署。每个节点应具有一个名为 fluent-bit-* 的 pod。

```
kubectl get pods -n amazon-cloudwatch
```

上述步骤在集群中创建了以下资源：

- amazon-cloudwatch 命名空间中名为 Fluent-Bit 的服务账户。该服务账户用于运行 Fluent Bit daemonSet。有关更多信息，请参阅《Kubernetes 参考》中的管理服务账户。
- amazon-cloudwatch 命名空间中名为 Fluent-Bit-role 的集群角色。该集群角色为 Fluent-Bit 服务账户授予有关 pod 日志的 get、list 和 watch 权限。有关更多信息，请参阅《Kubernetes 参考》中的 API 概述。
- amazon-cloudwatch 命名空间中的名为 Fluent-Bit-config 的 ConfigMap。该 ConfigMap 包含由 Fluent Bit 使用的配置。有关更多信息，请参阅《Kubernetes 任务》文档中的配置 Pod 以使用 ConfigMap。

如果要验证 Fluent Bit 设置，请按照下列步骤操作。

验证 Fluent Bit 设置

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Log groups（日志组）。
3. 确保您位于部署 Fluent Bit 的区域中。
4. 检查区域中的日志组列表。您将看到以下内容：
   - /aws/containerinsights/*Cluster_Name*/application
   - /aws/containerinsights/*Cluster_Name*/host

- /aws/containerinsights/*Cluster_Name*/dataplane

5. 导航到其中一个日志组，并检查日志流的 Last Event Time（上次事件时间）。如果相对于您部署 Fluent Bit 的时间，它是最近的，则会验证设置。

   创建 /dataplane 日志组时，可能会稍有延迟。这是正常的，因为只有在 Fluent Bit 开始为该日志组发送日志时才会创建这些日志组。

多行日志支持

有关如何将 Fluent Bit 用于多行日志的信息，请参阅 Fluent Bit 文档的以下部分：

- 多行解析
- 多行和容器（v1.8）
- 多行核心（v1.8）
- 始终在尾部输入中使用多行

（可选）从 Fluent Bit 减少日志卷

默认情况下，我们将 Fluent Bit 应用程序日志和 Kubernetes 元数据发送到 CloudWatch。如果您要减少发送到 CloudWatch 的数据量，可以停止将这些数据源中的一个或两个发送到 CloudWatch。如果您已按照本页上的步骤设置 Fluent Bit，则请通过之前运行的 kubectl apply 命令下载 Kubernetes 清单 YAML 文件，并根据您的更改对其进行修改，然后可以将其重新应用到集群。或者，如果您使用的是 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表，则请参阅（可选）其他配置，了解有关使用加载项的高级配置或 Helm 图表管理 Fluent Bit 配置的信息。

要停止 Fluent Bit 应用程序日志，请从 Fluent Bit configuration 文件中删除以下部分。

```
[INPUT]
        Name                tail
        Tag                 application.*
        Path                /var/log/containers/fluent-bit*
        Parser              docker
        DB                  /fluent-bit/state/flb_log.db
        Mem_Buf_Limit       5MB
        Skip_Long_Lines     On
        Refresh_Interval    10
```

要取消将 Kubernetes 元数据附加到发送到 CloudWatch 的日志事件，请在 Fluent Bit 配置的 `application-log.conf` 部分中添加以下筛选条件。将 *<Metadata_1>* 和类似字段替换为实际的元数据标识符。

```
application-log.conf: |
    [FILTER]
        Name                nest
        Match               application.*
        Operation           lift
        Nested_under        kubernetes
        Add_prefix          Kube.

    [FILTER]
        Name                modify
        Match               application.*
        Remove              Kube.<Metadata_1>
        Remove              Kube.<Metadata_2>
        Remove              Kube.<Metadata_3>

    [FILTER]
        Name                nest
        Match               application.*
        Operation           nest
        Wildcard            Kube.*
        Nested_under        kubernetes
        Remove_prefix       Kube.
```

故障排除

如果您没有看到这些日志组并且查看的是正确区域，请检查 Fluent Bit daemonSet pod 日志以查找错误。

运行以下命令，并确保状态为 `Running`。

```
kubectl get pods -n amazon-cloudwatch
```

如果日志具有与 IAM 权限相关的错误，请检查附加到集群节点的 IAM 角色。有关运行 Amazon EKS 集群所需权限的更多信息，请参阅 Amazon EKS 用户指南中的 Amazon EKS IAM 策略、角色和权限。

如果 pod 状态为 `CreateContainerConfigError`，请运行以下命令以获取确切的错误。

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

控制面板

您可以创建一个控制面板来监控每个正在运行的插件的指标。您可以查看输入和输出字节以及记录处理率以及输出错误和重试/失败率的数据。要查看这些指标，您需要为 Amazon EKS 和 Kubernetes 集群安装带有 Prometheus 指标集合的 CloudWatch 代理。有关如何设置控制面板的更多信息，请参阅 在 Amazon EKS 和 Kubernetes 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理。

> **Note**
>
> 在设置此控制面板之前，您必须为 Prometheus 指标设置 Container Insights。有关更多信息，请参阅 Container Insights Prometheus 指标监控。

为 Fluent Bit Prometheus 指标创建控制面板

1. 创建环境变量，替换以下行中右侧的值以匹配您的部署。

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-west-1
CLUSTER_NAME=your_kubernetes_cluster_name
```

2. 通过运行以下命令来创建控制面板。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --
dashboard-body
```

（可选）将 Fluentd 设置为 DaemonSet 以将日志发送到 CloudWatch Logs

> **Warning**
>
> Container Insights 对 Fluentd 的支持目前处于维护模式，这意味着 AWS 不会为 Fluentd 提供任何进一步的更新，我们计划在不久的将来将其弃用。此外，Container Insights 的

当前 Fluentd 配置使用的是旧版本的 Fluentd 映像 `fluent/fluentd-kubernetes-daemonset:v1.10.3-debian-cloudwatch-1.0`，该版本没有最新的改进和安全补丁。有关开源社群支持的最新 Fluentd 映像，请参阅 fluentd-kubernetes-daemonset。
如果可能，我们强烈建议您迁移以将 FluentBit 与 Container Insights 结合使用。使用 FluentBit 作为 Container Insights 的日志转发器，可显着提高性能。
有关更多信息，请参阅 将 Fluent Bit 设置为 DaemonSet 以将日志发送到 CloudWatch Logs 和 您在使用 Fluentd 时出现的差异。

要设置 Fluentd 以从容器中收集日志，您可以按照 Amazon EKS 和 Kubernetes 上的 Container Insights 的快速入门设置 中的步骤操作，也可以按照本节中的步骤操作。在以下步骤中，您将 Fluentd 设置为 DaemonSet 以将日志发送到 CloudWatch Logs。在完成该步骤时，Fluentd 将创建以下日志组（如果日志组尚不存在）。

日志组名称	日志源
`/aws/containerinsights/Cluster_Name/application`	`/var/log/containers` 中的所有日志文件
`/aws/containerinsights/Cluster_Name/host`	`/var/log/dmesg`、`/var/log/secure` 和 `/var/log/messages` 中的日志
`/aws/containerinsights/Cluster_Name/dataplane`	`kubelet.service`、`kubeproxy.service` 和 `docker.service` 的 `/var/log/journal` 中的日志。

步骤 1：为 CloudWatch 创建命名空间

使用以下步骤为 CloudWatch 创建名为 `amazon-cloudwatch` 的 Kubernetes 命名空间。如果已创建该命名空间，您可以跳过此步骤。

为 CloudWatch 创建命名空间

- 输入以下 命令。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

步骤 2：安装 Fluentd

下载 Fluentd 以开始执行该过程。在完成这些步骤时，部署将在集群上创建以下资源：

- `amazon-cloudwatch` 命名空间中名为 `fluentd` 的服务账户。该服务账户用于运行 Fluentd DaemonSet。有关更多信息，请参阅《Kubernetes 参考》中的 管理服务账户。
- `amazon-cloudwatch` 命名空间中名为 `fluentd` 的集群角色。该集群角色为 `fluentd` 服务账户授予有关 pod 日志的 `get`、`list` 和 `watch` 权限。有关更多信息，请参阅《Kubernetes 参考》中的 API 概述。
- `amazon-cloudwatch` 命名空间中的名为 `fluentd-config` 的 ConfigMap。该 ConfigMap 包含由 Fluentd 使用的配置。有关更多信息，请参阅《Kubernetes 任务》文档中的 配置 Pod 以使用 ConfigMap。

要安装 Fluentd

1. 使用集群名称和日志将发送到的 AWS 区域创建一个名为 `cluster-info` 的 ConfigMap。运行以下命令，并使用您的集群和区域名称更新占位符。

```
kubectl create configmap cluster-info \
--from-literal=cluster.name=cluster_name \
--from-literal=logs.region=region_name -n amazon-cloudwatch
```

2. 运行以下命令将 Fluentd DaemonSet 下载并部署到集群中。确保您使用的是具有正确架构的容器镜像。示例清单仅适用于 x86 实例，并将输入 `CrashLoopBackOff`（如果您的集群中有高级 RISC 机器 (ARM) 实例）。Fluentd daemonSet 没有官方的多架构 Docker 映像，不允许您对多个底层镜像使用一个标签，并让容器运行时系统拉取正确标签。Fluentd ARM 映像使用带有 `arm64` 后缀的不同标签。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/fluentd/fluentd.yaml
```

> **Note**
>
> 由于最近进行了更改，以优化 Fluentd 配置并最大限度地减少 Fluentd API 请求对 Kubernetes API 端点的影响，因此默认情况下，Kubernetes 筛选条件的“监视”选项已禁用。有关更多详细信息，请参阅 fluent-plugin-kubernetes_metadata_filter。

3. 运行以下命令以验证部署。每个节点应具有一个名为 `fluentd-cloudwatch-*` 的 pod。

```
kubectl get pods -n amazon-cloudwatch
```

步骤 3：验证 Fluentd 设置

要验证您的 Fluentd 设置，请按照以下步骤进行操作。

要验证 Container Insights 的 Fluentd 设置

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Log groups（日志组）。确保您位于将 Fluentd 部署到您的容器的区域中。

   在该区域上的日志组列表中，您将会看到以下内容：

   - `/aws/containerinsights/`*`Cluster_Name`*`/application`
   - `/aws/containerinsights/`*`Cluster_Name`*`/host`
   - `/aws/containerinsights/`*`Cluster_Name`*`/dataplane`

   如果看到这些日志组，则 Fluentd 设置已验证完毕。

多行日志支持

2019 年 8 月 19 日，我们为 Fluentd 采集的日志增加了多行日志支持。

默认情况下，多行日志条目的起始字符为任何不包含空格的字符。这意味着，所有起始字符不含空格的日志行都将被视为新的多行日志条目。

如果您自己的应用程序日志使用其他多行起始字符规则，则可以通过在 `fluentd.yaml` 文件中进行两项更改来支持。

首先，通过将日志文件的路径名添加到 `fluentd.yaml` 的 `containers` 部分的 `exclude_path` 字段中，从而将其从默认的多行支持中排除。示例如下：

```
<source>
  @type tail
  @id in_tail_container_logs
  @label @containers
```

```
  path /var/log/containers/*.log
  exclude_path ["full_pathname_of_log_file*", "full_pathname_of_log_file2*"]
```

接下来，将日志文件的数据块添加到 fluentd.yaml 文件中。在以下示例中，CloudWatch 代理的日志文件使用时间戳正则表达式来作为多行起始字符规则。您可以复制此数据块并将其添加到 fluentd.yaml。更改指示的行，以反映您要使用的应用程序日志文件名和多行起始字符规则。

```
<source>
  @type tail
  @id in_tail_cwagent_logs
  @label @cwagentlogs
  path /var/log/containers/cloudwatch-agent*
  pos_file /var/log/cloudwatch-agent.log.pos
  tag *
  read_from_head true
 <parse>
    @type json
    time_format %Y-%m-%dT%H:%M:%S.%NZ
 </parse>
</source>
```

```
<label @cwagentlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_cwagent
  </filter>

  <filter **>
    @type record_transformer
    @id filter_cwagent_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
    </record>
 </filter>

  <filter **>
    @type concat
    key log
    multiline_start_regexp /^\d{4}[-/]\d{1,2}[-/]\d{1,2}/
    separator ""
```

```
    flush_interval 5
    timeout_label @NORMAL
 </filter>

 <match **>
    @type relabel
    @label @NORMAL
 </match>
</label>
```

（可选）从 Fluentd 减少日志卷

默认情况下，我们将 Fluentd 应用程序日志和 Kubernetes 元数据发送到 CloudWatch。如果您要减少发送到 CloudWatch 的数据量，可以停止将这些数据源中的一个或两个发送到 CloudWatch。

要停止 Fluentd 应用程序日志，请从 `fluentd.yaml` 文件中删除以下部分。

```
<source>
  @type tail
  @id in_tail_fluentd_logs
  @label @fluentdlogs
  path /var/log/containers/fluentd*
  pos_file /var/log/fluentd.log.pos
  tag *
  read_from_head true
   <parse>
   @type json
   time_format %Y-%m-%dT%H:%M:%S.%NZ
  </parse>
</source>

<label @fluentdlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_fluentd
  </filter>

  <filter **>
    @type record_transformer
    @id filter_fluentd_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
```

```
    </record>
   </filter>

   <match **>
     @type relabel
     @label @NORMAL
   </match>
</label>
```

要取消将 Kubernetes 元数据附加到发送到 CloudWatch 的日志事件，请在 `fluentd.yaml` 文件的 `record_transformer` 部分中添加一行。在要删除此元数据的日志源中，添加以下行。

```
remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
 $.kubernetes.container_image_id, $.kubernetes.namespace_id
```

例如：

```
<filter **>
  @type record_transformer
  @id filter_containers_stream_transformer
  <record>
    stream_name ${tag_parts[3]}
  </record>
  remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
 $.kubernetes.container_image_id, $.kubernetes.namespace_id
</filter>
```

故障排除

如果您没有看到这些日志组并且查看的是正确区域，请检查 Fluentd DaemonSet 容器组（pod）日志以查找错误。

运行以下命令，并确保状态为 `Running`。

```
kubectl get pods -n amazon-cloudwatch
```

在上一命令的结果中，记下以 `fluentd-cloudwatch` 开头的 pod 名称。在以下命令中使用该 pod 名称。

```
kubectl logs pod_name -n amazon-cloudwatch
```

如果日志具有与 IAM 权限相关的错误，请检查附加到集群节点的 IAM 角色。有关运行 Amazon EKS 集群所需权限的更多信息，请参阅 Amazon EKS 用户指南中的 Amazon EKS IAM 策略、角色和权限。

如果 pod 状态为 `CreateContainerConfigError`，请运行以下命令以获取确切的错误。

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

如果 pod 状态为 `CrashLoopBack0ff`，请确保 Fluentd 容器镜像的架构与您安装 Fluentd 时的节点相同。如果您的集群同时具有 x86 和 ARM64 节点，您可以使用 kubernetes.io/arch 标签将镜像放置在正确的节点上。有关更多信息，请参阅 kuberntes.io/arch。

（可选）设置 Amazon EKS 控制面板日志记录

如果使用的是 Amazon EKS，您可以选择启用 Amazon EKS 控制面板日志记录，以直接从 Amazon EKS 控制面板中向 CloudWatch Logs 提供审计和诊断日志。有关更多信息，请参阅 Amazon EKS 控制面板日志记录。

（可选）启用 App Mesh Envoy 访问日志

您可以设置 Container Insights Fluentd 以将 App Mesh Envoy 访问日志发送到 CloudWatch Logs。有关更多信息，请参阅日志记录。

将 Envoy 访问日志发送到 CloudWatch Logs

1. 在集群中设置 Fluentd。有关更多信息，请参阅（可选）将 Fluentd 设置为 DaemonSet 以将日志发送到 CloudWatch Logs。
2. 为您的虚拟节点配置 Envoy 访问日志。有关说明，请参阅日志记录。请确保在每个虚拟节点中将日志路径配置为 **`/dev/stdout`**。

完成后，Envoy 访问日志将发送到 `/aws/containerinsights/Cluster_Name/application` 日志组。

（可选）为大型集群启用 Use_Kubelet 功能

默认情况下，在 FluentBit Kubernetes 插件中禁用 Use_Kubelet 功能。启用此功能可以减少 API 服务器的流量，并缓解 API 服务器成为瓶颈的问题。我们建议您为大型集群启用此功能。

要启用 Use_Kubelet，请先将节点和节点/代理权限添加到 clusterRole config 中。

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
      - /metrics
    verbs:
      - get
  - apiGroups: [""]
    resources:
      - namespaces
      - pods
      - pods/logs
      - nodes
      - nodes/proxy
    verbs: ["get", "list", "watch"]
```

在 DaemonSet 配置中，此功能需要主机网络访问权限。适用于 amazon/aws-for-fluent-bit 的镜像版本应为 2.12.0 或更高版本，或者 fluent bit 镜像版本应为 1.7.2 或更高版本。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
      - name: fluent-bit
        image: amazon/aws-for-fluent-bit:2.19.0
```

```
        imagePullPolicy: Always
        env:
            - name: AWS_REGION
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: logs.region
            - name: CLUSTER_NAME
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: cluster.name
            - name: HTTP_SERVER
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: http.server
            - name: HTTP_PORT
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: http.port
            - name: READ_FROM_HEAD
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: read.head
            - name: READ_FROM_TAIL
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: read.tail
            - name: HOST_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: HOSTNAME
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: metadata.name
            - name: CI_VERSION
              value: "k8s/1.3.8"
        resources:
```

```
            limits:
              memory: 200Mi
            requests:
              cpu: 500m
              memory: 100Mi
        volumeMounts:
        # Please don't change below read-only permissions
        - name: fluentbitstate
          mountPath: /var/fluent-bit/state
        - name: varlog
          mountPath: /var/log
          readOnly: true
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: fluent-bit-config
          mountPath: /fluent-bit/etc/
        - name: runlogjournal
          mountPath: /run/log/journal
          readOnly: true
        - name: dmesg
          mountPath: /var/log/dmesg
          readOnly: true
      terminationGracePeriodSeconds: 10
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      volumes:
      - name: fluentbitstate
        hostPath:
          path: /var/fluent-bit/state
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
      - name: fluent-bit-config
        configMap:
          name: fluent-bit-config
      - name: runlogjournal
        hostPath:
          path: /run/log/journal
      - name: dmesg
        hostPath:
```

```
          path: /var/log/dmesg
      serviceAccountName: fluent-bit
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      - operator: "Exists"
        effect: "NoExecute"
      - operator: "Exists"
        effect: "NoSchedule"
```

Kubernetes 插件配置应该类似于以下内容：

```
[FILTER]
        Name                kubernetes
        Match               application.*
        Kube_URL            https://kubernetes.default.svc:443
        Kube_Tag_Prefix     application.var.log.containers.
        Merge_Log           On
        Merge_Log_Key       log_processed
        K8S-Logging.Parser  On
        K8S-Logging.Exclude Off
        Labels              Off
        Annotations         Off
        Use_Kubelet         On
        Kubelet_Port        10250
        Buffer_Size         0
```

在 Amazon EKS 和 Kubernetes 上更新或删除 Container Insights

使用这些章节中的步骤更新 CloudWatch 代理容器镜像，或从 Amazon EKS 或 Kubernetes 集群中删除 Container Insights。

主题

- 升级到针对 Amazon EKS 增强了可观测性的 Container Insights
- 更新 CloudWatch 代理容器镜像
- 删除 Container Insights 的 CloudWatch 代理和 Fluent Bit

升级到针对 Amazon EKS 增强了可观测性的 Container Insights

> **Important**
>
> 如果您要在 Amazon EKS 集群上升级或安装 Container Insights，则建议您使用 Amazon CloudWatch Observability EKS 附加组件进行安装，而不是按照本部分中的说明进行安装。此外，要检索加速计算指标，必须使用 Amazon CloudWatch Observability EKS 附加组件。有关更多信息和说明，请参阅 安装 Amazon CloudWatch Observability EKS 附加组件。

针对 Amazon EKS 增强了可观测性的 Container Insights 是 Container Insights 的最新版本。它从运行 Amazon EKS 的集群收集详细指标，并提供精选的、可立即使用的控制面板，以深入了解应用程序和基础设施的遥测数据。有关此版本的 Container Insights 的更多信息，请参阅 针对 Amazon EKS 增强了可观测性的 Container Insights。

如果您已在 Amazon EKS 集群中安装了 Container Insights 的原始版本，并且想要将其升级到增强了可观测性的较新版本，请按照本部分中的说明进行操作。

> **Important**
>
> 在完成本节中的步骤之前，您必须已对包含 cert-manager 等先决条件进行验证。有关更多信息，请参阅 使用 CloudWatch 代理 operator 和 Fluent Bit 快速入门。

将 Amazon EKS 集群升级到针对 Amazon EKS 增强了可观测性的 Container Insights

1. 输入以下命令以安装 CloudWatch 代理 operator。将 *my-cluster-name* 替换为 Amazon EKS 或 Kubernetes 集群的名称，将 *my-cluster-region* 替换为在其中发布日志的区域的名称。我们建议您使用在其中部署集群的同一个区域来降低 AWS 出站数据传输成本。

   ```
   ClusterName=my-cluster-name
   RegionName=my-cluster-region
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
   container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
   {{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl
    apply -f -
   ```

   如果您发现由于资源冲突而导致故障，则可能是因为您已在集群上安装了 CloudWatch 代理和 Fluent Bit 及其关联组件（例如 ServiceAccount、ClusterRole 和 ClusterRoleBinding）。当

CloudWatch 代理 operator 尝试安装 CloudWatch 代理及其关联组件时，如果检测到内容有任何变化，则在默认情况下这会使安装或更新失败，以避免覆盖集群上资源的状态。建议您删除之前在集群上安装的带有 Container Insights 设置的所有现有 CloudWatch 代理，然后安装 CloudWatch 代理 operator。

2. （可选）要应用现有的自定义 Fluent Bit 配置，必须更新与 Fluent Bit 守护程序集关联的 configmap。CloudWatch 代理 operator 为 Fluent Bit 提供了默认配置，您可以根据需要覆盖或修改默认配置。要应用自定义配置，请按照以下步骤操作。

   a. 通过输入以下命令打开现有配置。

   ```
   kubectl edit cm fluent-bit-config -n amazon-cloudwatch
   ```

   b. 在文件中进行更改，然后输入 `:wq` 保存文件并退出编辑模式。

   c. 通过输入以下命令重启 Fluent Bit。

   ```
   kubectl rollout restart fluent-bit -n amazon-cloudwatch
   ```

更新 CloudWatch 代理容器镜像

> **Important**
>
> 如果您要在 Amazon EKS 集群上升级或安装 Container Insights，则建议您使用 Amazon CloudWatch Observability EKS 附加组件进行安装，而不是按照本部分中的说明进行安装。此外，要检索加速计算指标，必须使用 Amazon CloudWatch Observability EKS 附加组件或 CloudWatch 代理 operator。有关更多信息和说明，请参阅 安装 Amazon CloudWatch Observability EKS 附加组件。

如果需要将容器映像更新到最新版本，请使用本节中的步骤。

更新您的容器映像

1. 通过输入以下命令，验证 `amazoncloudwatchagent` 客户资源定义（CRD）是否已经存在。

   ```
   kubectl get crds amazoncloudwatchagents.cloudwatch.aws.amazon.com -n amazon-
   cloudwatch
   ```

如果此命令返回缺少 CRD 的错误，则集群没有使用 CloudWatch 代理 operator 配置针对 Amazon EKS 增强了可观测性的 Container Insights。在这种情况下，请参阅升级到针对 Amazon EKS 增强了可观测性的 Container Insights。

2. 输入以下命令，应用最新的 `cwagent-version.yaml` 文件。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-version.yaml | kubectl apply -f -
```

删除 Container Insights 的 CloudWatch 代理和 Fluent Bit

如果您通过安装适用于 Amazon EKS 的 CloudWatch Observability 附加组件来安装 Container Insights，则可以通过输入以下命令删除 Container Insights 和 CloudWatch 代理：

> **Note**
>
> Amazon EKS 附加组件现在支持 Windows Worker 节点上的 Container Insights。如果您删除 Amazon EKS 附加组件，则 Container Insights for Windows 也会被删除。

```
aws eks delete-addon –cluster-name my-cluster –addon-name amazon-cloudwatch-observability
```

否则，要删除与 CloudWatch 代理和 Fluent Bit 相关的所有资源，请输入以下命令。在此命令中，*My_Cluster_Name* 是 Amazon EKS 或 Kubernetes 集群的名称，*My_Region* 是在其中发布日志的区域的名称。

```
ClusterName=My_Cluster_Name
RegionName=My-Region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl delete -f -
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete -f -
```

查看 Container Insights 指标

在设置了 Container Insights 并开始收集指标后，您可以在 CloudWatch 控制台中查看这些指标。

要在控制面板上显示 Container Insights 指标，您必须完成 Container Insights 设置。有关更多信息，请参阅 设置 Container Insights。

此过程说明如何查看 Container Insights 根据收集的日志数据自动生成的指标。本节的其余部分介绍了如何进一步深入了解您的数据并使用 CloudWatch Logs Insights 查看更多精细级别的指标。

查看 Container Insights 指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Insights、Container Insights。
3. 在 Container Insights 下的下拉框中，选择性能监控。
4. 使用靠近顶部的下拉框选择要查看的资源类型以及特定资源。

您可以设置 Container Insights 收集的任何指标的 CloudWatch 告警。有关更多信息，请参阅 使用 Amazon CloudWatch 告警

> **Note**
>
> 如果您已设置 CloudWatch Application Insights 来监控容器化应用程序，Application Insights 控制面板将显示在 Container Insights 控制面板下方。如果尚未启用 Application Insights，可以通过在 Container Insights 控制面板中的性能视图下方选择 Auto-configure Application Insights（自动配置 Application Insights）将其启用。
> 有关 Application Insights 和容器化应用程序的更多信息，请参阅 为 Amazon ECS 和 Amazon EKS 资源监控启用 Application Insights。

查看排名靠前的贡献者

对于 Container Insights 性能监控中的某些视图，您还可以通过内存或 CPU 或最近的活动资源查看排名靠前的贡献者。当您在靠近页面顶部的下拉框中选择以下任一控制面板时，此功能可用：

- ECS 服务
- ECS 任务

- EKS 命名空间
- EKS 服务
- EKS Pods

当您查看这些类型的任一资源时，页面底部会显示一个最初按 CPU 使用率排序的表格。您可以将其更改为按内存使用情况或最近的活动进行排序。要查看有关表格中某行的详细信息，可以选中该行旁边的复选框，然后选择 Actions（操作），然后选择 Actions（操作）菜单中的任一选项。

使用 CloudWatch Logs Insights 查看 Container Insights 数据

Container Insights 通过使用嵌入式指标格式的性能日志事件来收集指标。日志存储在 CloudWatch Logs 中。CloudWatch 从日志中自动生成多个指标，您可以在 CloudWatch 控制台中查看这些指标。您还可以对使用 CloudWatch Logs Insights 查询收集到的性能数据进行更深入的分析。

有关 CloudWatch Logs Insights 的更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。有关您可以在查询中使用的日志字段的更多信息，请参阅 Amazon EKS 和 Kubernetes 的 Container Insights 性能日志事件。

使用 CloudWatch Logs Insights 查询容器指标数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Insights。

   屏幕顶部附近是查询编辑器。当您首次打开 CloudWatch Logs Insights 时，此框包含一个默认查询，该查询将返回 20 个最新的日志事件。
3. 在查询编辑器上方的框中，选择要查询的 Container Insights 日志组之一。要成功执行以下示例查询，日志组名称必须以 performance 结尾。

   当您选择日志组时，CloudWatch Logs Insights 会自动检测日志组数据中的字段，并将其显示在右侧窗格中的 Discovered fields（发现的字段）中。它还显示此日志组中的日志事件随时间变化的条形图。该条形图显示与您的查询和时间范围匹配的日志组中的事件分布情况，而不仅仅是表中显示的事件。
4. 在查询编辑器中，将默认查询替换为以下查询，然后选择运行查询。

   ```
   STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
   | SORT avg_node_cpu_utilization DESC
   ```

   该查询显示一个节点列表，它按平均节点 CPU 使用率进行排序。

5. 要尝试使用另一个示例，请将该查询替换为另一个查询，然后选择运行查询。本页稍后将列出更多示例查询。

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by
 PodName
| SORT avg_number_of_container_restarts DESC
```

   该查询显示一个 pod 列表，它按平均容器重新启动次数进行排序。

6. 如果要尝试使用另一个查询，您可以使用屏幕右侧的列表中的包含字段。有关查询语法的更多信息，请参阅 CloudWatch Logs Insights 查询语法。

查看资源列表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择资源。
3. 默认视图是由 Container Insights 监控的资源列表，以及您在这些资源上设置的警报。要查看资源的可视映射，请选择地图视图。
4. 在地图视图中，您可以将指针悬停在地图中的任何资源上，查看有关该资源的基本指标。您可以选择任意资源来查看有关该资源的详细图形。

使用案例：查看 Amazon ECS 容器中的任务级指标

以下示例说明了如何使用 CloudWatch Logs Insights 深入了解 Container Insights 日志。有关更多示例，请参阅博客推出适用于 Amazon ECS 的 Amazon CloudWatch Container Insights。

Container Insights 不会在精细的任务级别自动生成指标。以下查询显示 CPU 和内存使用率的任务级指标。

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName
| sort Mem, CPU desc
```

Container Insights 的其他示例查询

pod 列表，按平均容器重新启动次数排序

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName
| SORT avg_number_of_container_restarts DESC
```

请求的 Pod 与运行的 pod

```
fields @timestamp, @message
| sort @timestamp desc
| filter Type="Pod"
| stats min(pod_number_of_containers) as requested,
 min(pod_number_of_running_containers) as running, ceil(avg(pod_number_of_containers-
pod_number_of_running_containers)) as pods_missing by kubernetes.pod_name
| sort pods_missing desc
```

集群节点故障计数

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures
| filter Type="Cluster"
| sort @timestamp desc
```

按容器名称的应用程序日志错误

```
stats count() as countoferrors by kubernetes.container_name
| filter stream="stderr"
| sort countoferrors desc
```

Container Insights 收集的指标

Container Insights 为 Amazon ECS 和 Amazon ECS 上的 AWS Fargate 收集一组指标，并为 Amazon EKS、Amazon EKS 上的 AWS Fargate 和 Kubernetes 收集另一组指标。

容器任务运行一段时间后，指标才可见。

主题

- Amazon ECS Container Insights 指标
- Amazon EKS 和 Kubernetes Container Insights 指标

Amazon ECS Container Insights 指标

下表列出了 Container Insights 收集的针对 Amazon ECS 的指标和维度。这些指标位于 `ECS/ContainerInsights` 命名空间中。有关更多信息，请参阅 指标。

如果您在控制台中未看到任何 Container Insights 指标，请确保已完成 Container Insights 的设置。在完全设置 Container Insights 之前，指标不会显示。有关更多信息，请参阅 设置 Container Insights。

完成在 Amazon ECS 上针对集群级别和服务级别指标设置 Container Insights中的步骤后，以下指标将可用

指标名称	Dimensions	描述
ContainerInstanceCount	ClusterName	注册到集群的运行 Amazon ECS 代理的 EC2 实例的数目。 仅针对在集群中运行 Amazon ECS 任务的容器实例收集此指标。对于没有任何 Amazon ECS 任务的空容器实例，不会收集此指标。 单位：计数
CpuUtilized	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	任务在您使用的维度集所指定的资源中使用的 CPU 单元。 仅针对在任务定义中具有已定义的 CPU 保留的任务收集此指标。 单位：无
CpuReserved	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	任务在您使用的维度集所指定的资源中预留的 CPU 单元。 仅针对在任务定义中具有已定义的 CPU 保留的任务收集此指标。 单位：无
DeploymentCount	ServiceName , ClusterName	Amazon ECS 服务中的部署数。

指标名称	Dimensions	描述
		单位：计数
DesiredTaskCount	ServiceName，ClusterName	Amazon ECS 服务所需的任务数。 单位：计数
EBSFilesystemSize	VolumeName，TaskDefinitionFamily，ClusterName TaskDefinitionFamily，ClusterName ServiceName，ClusterName	分配给您使用的维度所指定资源的 Amazon EBS 文件系统存储总量，以千兆字节（GB）为单位。 该指标仅适用于使用平台版本 1.4.0 在 Fargate 上运行的 Amazon ECS 基础设施上运行的任务，或在使用容器代理版本 1.79.0 或更高版本的 Amazon EC2 实例上运行的任务。 单位：千兆字节（GB）

指标名称	Dimensions	描述
EBSFilesystemUtilized	VolumeName，TaskDefinitionFamily，ClusterName TaskDefinitionFamily，ClusterName ServiceName，ClusterName	您使用的维度所指定资源使用的 Amazon EBS 文件系统存储总量，以千兆字节（GB）为单位。 该指标仅适用于使用平台版本 1.4.0 在 Fargate 上运行的 Amazon ECS 基础设施上运行的任务，或在使用容器代理版本 1.79.0 或更高版本的 Amazon EC2 实例上运行的任务。 对于在 Fargate 上运行的任务，Fargate 会在磁盘上预留仅供 Fargate 使用的空间。Fargate 使用的空间不会产生任何成本，但可以使用类似 df 的工具看到额外的存储空间。 单位：千兆字节（GB）

指标名称	Dimensions	描述
EphemeralStorageReserved [1]	TaskDefinitionFamily ，ClusterName ServiceName ，ClusterName ClusterName	从您使用的维度所指定的资源中临时存储保留的字节数。临时存储用于容器根文件系统以及容器映像和任务定义中所定义的任何绑定装载主机卷。无法在正在运行的任务中更改临时存储量。 该指标仅适用于在 Fargate Linux 平台 1.4.0 或更高版本上运行的任务。 单位：千兆字节（GB）
EphemeralStorageUtilized [1]	TaskDefinitionFamily ，ClusterName ServiceName ，ClusterName ClusterName	从您使用的维度所指定的资源中临时存储使用的字节数。临时存储用于容器根文件系统以及容器映像和任务定义中所定义的任何绑定装载主机卷。无法在正在运行的任务中更改临时存储量。 该指标仅适用于在 Fargate Linux 平台 1.4.0 或更高版本上运行的任务。 单位：千兆字节（GB）

指标名称	Dimensions	描述
MemoryUtilized	TaskDefinitionFamily ，ClusterName ServiceName ，ClusterName ClusterName	任务在您使用的维度集所指定的资源中使用的内存。 仅针对在任务定义中具有已定义的内存保留的任务收集此指标。 单位：兆字节
MemoryReserved	TaskDefinitionFamily ，ClusterName ServiceName ，ClusterName ClusterName	任务在您使用的维度集所指定的资源中预留的内存。 仅针对在任务定义中具有已定义的内存保留的任务收集此指标。 单位：兆字节
NetworkRxBytes	TaskDefinitionFamily ，ClusterName ServiceName ，ClusterName ClusterName	您使用的维度所指定的资源接收的字节数。该指标是从 Docker 运行时系统获取的。 此指标仅适用于使用 awsvpc 或 bridge 网络模式的任务中的容器。 单位：字节/秒

指标名称	Dimensions	描述
NetworkTxBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	您使用的维度所指定的资源传输的字节数。该指标是从 Docker 运行时系统获取的。 此指标仅适用于使用 awsvpc 或 bridge 网络模式的任务中的容器。 单位：字节/秒
PendingTaskCount	ServiceName , ClusterName	当前处于 PENDING 状态的任务的数量。 单位：计数
RunningTaskCount	ServiceName , ClusterName	当前处于 RUNNING 状态的任务的数量。 单位：计数
RestartCount	ServiceName , TaskDefinitionFamily	Amazon ECS 任务中容器重新启动的次数。 仅会对启用了重启策略的容器收集此指标。 单位：计数
ServiceCount	ClusterName	集群中的服务数。 单位：计数

指标名称	Dimensions	描述
StorageReadBytes	TaskDefinitionFamily，ClusterName ServiceName，ClusterName ClusterName	从您使用之维度所指定资源中实例上的存储读取的字节数。这不包括存储设备的读取字节数。该指标是从 Docker 运行时系统获取的。 单位：字节
StorageWriteBytes	TaskDefinitionFamily，ClusterName ServiceName，ClusterName ClusterName	写入到您使用的维度所指定的资源中的存储的字节数。该指标是从 Docker 运行时系统获取的。 单位：字节
TaskCount	ClusterName	正在集群中运行的任务数。 单位：计数
TaskSetCount	ServiceName，ClusterName	服务中设置的任务数。 单位：计数

> **Note**
>
> EphemeralStorageReserved 和 EphemeralStorageUtilized 指标仅适用于在 Fargate Linux 平台 1.4.0 或更高版本上运行的任务。
> Fargate 可保留磁盘空间。该磁盘空间仅由 Fargate 使用。您无需为此付费。它没有显示在这些指标中。但是，您可以在 df 等其他工具中看到这种额外的存储空间。

完成[部署 CloudWatch 代理以收集 Amazon ECS 上的 EC2 实例级别指标](#)中的步骤后，以下指标将可用

指标名称	Dimensions	描述
instance_cpu_limit	ClusterName	可分配给集群中的单个 EC2 实例的 CPU 单元的最大数目。 单位：无
instance_cpu_reserved_capacity	ClusterName InstanceId，ContainerInstanceId，ClusterName	集群中单个 EC2 实例上当前预留的 CPU 的百分比。 单位：百分比
instance_cpu_usage_total	ClusterName	集群中单个 EC2 实例上正在使用的 CPU 单元的数目。 单位：无
instance_cpu_utilization	ClusterName InstanceId，ContainerInstanceId，ClusterName	集群中单个 EC2 实例上正在使用的 CPU 单元的总百分比。 单位：百分比
instance_filesystem_utilization	ClusterName InstanceId，ContainerInstanceId，ClusterName	集群中单个 EC2 实例上正在使用的文件系统容量的总百分比。 单位：百分比
instance_memory_limit	ClusterName	可分配给此集群中单个 EC2 实例的最大内存量（以字节为单位）。 单位：字节

指标名称	Dimensions	描述
instance_memory_reserved_capacity	ClusterName InstanceId ，ContainerInstanceId ，ClusterName	集群中单个 EC2 实例上当前预留的内存的百分比。 单位：百分比
instance_memory_utilization	ClusterName InstanceId ，ContainerInstanceId ，ClusterName	集群中单个 EC2 实例上正在使用的内存的总百分比。 单位：百分比
instance_memory_working_set	ClusterName	集群中单个 EC2 实例上正在使用的内存量（以字节为单位）。 单位：字节
instance_network_total_bytes	ClusterName	集群中单个 EC2 实例通过网络传输和接收的每秒总字节数。 单位：字节/秒
instance_number_of_running_tasks	ClusterName	集群中单个 EC2 实例上正在运行的任务的数目。 单位：计数

Amazon EKS 和 Kubernetes Container Insights 指标

下表列出了 Container Insights 为 Amazon EKS 和 Kubernetes 收集的指标和维度。这些指标位于 ContainerInsights 命名空间中。有关更多信息，请参阅 指标。

如果您在控制台中未看到任何 Container Insights 指标，请确保已完成 Container Insights 的设置。在完全设置 Container Insights 之前，指标不会显示。有关更多信息，请参阅 设置 Container Insights。

如果您使用 Amazon EKS 附加组件 1.5.0 版或更高版本或者 CloudWatch 代理 1.300035.0 版，则将为 Linux 和 Windows 节点收集下表中列出的大多数指标。请参阅该表的指标名称列，了解哪些指标并非为 Windows 收集。

在 Container Insights 的原始版本中，这些指标将作为自定义指标收费。借助针对 Amazon EKS 增强了可观测性的 Container Insights，Container Insights 指标按每次观测收费，而不是按存储的指标或摄取的日志收费。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> 在 Windows 上，不会为主机进程容器收集 `pod_network_rx_bytes` 和 `pod_network_tx_bytes` 等网络指标。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
`cluster_failed_node_count`	`ClusterName`		集群中失败的工作线程节点的数目。如果节点遭受任何节点条件的影响，则该节点被视为失败。有关更多信息，请参阅 Kubernetes 文档中的条件。
`cluster_node_count`	`ClusterName`		集群中工作线程节点的总数。

`namespace_number_of_running_pods`	`Namespace` `ClusterName`
	
	`ClusterName`
			您使用的维度所指定的资源中每个命名空间运行的 pod 的数目。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_cpu_limit	ClusterName	ClusterName，InstanceId，NodeName	可以分配给此集群中单个节点的 CPU 单元的最大数目。
node_cpu_reserved_capacity	NodeName, ClusterName，InstanceId ClusterName		为节点组件保留的 CPU 单元的百分比，例如 kubelet、kube-proxy 和 Docker。 公式：node_cpu_request / node_cpu_limit **Note** node_cpu_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。
node_cpu_usage_total	ClusterName	ClusterName，InstanceId，NodeName	集群中节点上正在使用的 CPU 单元的数目。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName		集群中节点上正在使用的 CPU 单元的总百分比。 公式：node_cpu_usage_total / node_cpu_limit

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_filesystem_utilization	NodeName, ClusterName , InstanceId ClusterName		集群中节点上正在使用的文件系统容量的总百分比。 公式：node_filesystem_usage / node_filesystem_capacity **Note** node_filesystem_usage 和 node_filesystem_capacity 不是直接作为指标报告，而是性能日志事件中的字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_memory_limit	ClusterName	ClusterName，InstanceId，NodeName	可以分配给此集群中单个节点的最大内存量（以字节为单位）。
node_filesystem_inodes 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights。其在 Windows 上不可用。		ClusterName ClusterName，InstanceId，NodeName	节点上 inode（已使用和未使用）的总数。
node_filesystem_inodes_free 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights。其在 Windows 上不可用。		ClusterName ClusterName，InstanceId，NodeName	节点上未使用 inode 的总数。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName		集群中节点上当前正在使用的内存百分比。 公式：node_memory_request / node_memory_limit **Note** node_memory_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName		一个或多个节点当前正在使用的内存百分比。它是节点内存使用量除以节点内存限制的百分比。 公式：node_memory_working_set / node_memory_limit 。
node_memory_working_set	ClusterName	ClusterName , InstanceId , NodeName	集群中节点的工作集中正在使用的内存量（以字节为单位）。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName		集群中每个节点通过网络传输和接收的每秒总字节数。 公式：node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes 和 node_network_tx_bytes 不是直接作为指标报告，而是性能日志事件中的字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName		集群中每个节点的正在运行的容器数。
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName		集群中每个节点上运行的 pod 的数量。
node_status_allocatable_pods 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName , InstanceId , NodeName	根据节点的可分配资源可以分配给节点的容器组（pod）数量，这定义为考虑系统进程守护程序预留和硬驱逐阈值后的节点容量余数。
node_status_capacity_pods 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName , InstanceId , NodeName	根据节点容量可以分配给节点的容器组（pod）数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_status_condition_ready 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	表示节点状态条件是 Ready 是否为 true。
node_status_condition_memory_pressure 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	表示节点状态条件是 MemoryPressure 是否为 true。
node_status_condition_pid_pressure 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	表示节点状态条件是 PIDPressure 是否为 true。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_status_condition_disk_pressure 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	表示节点状态条件是 OutOfDisk 是否为 true。
node_status_condition_unknown 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	表示是否有任何节点状态条件为“未知”。
node_interface_network_rx_dropped 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	节点上的网络接口接收并随后丢弃的数据包数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
node_interface_network_tx_dropped 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，InstanceId，NodeName	本应传输但被节点上的网络接口丢弃的数据包数量。
node_diskio_io_service_bytes_total 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights。其在 Windows 上不可用。		ClusterName ClusterName，InstanceId，NodeName	节点上所有 I/O 操作传输的总字节数。
node_diskio_io_serviced_total 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights。其在 Windows 上不可用。		ClusterName ClusterName，InstanceId，NodeName	节点上 I/O 操作的总数。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_cpu_reserved_capacity	PodName、Namespace、ClusterName ClusterName	ClusterName ，Namespace ，PodName, FullPodName ClusterName ，Namespace ，Service	集群中每个 pod 预留的 CPU 容量。 公式：pod_cpu_request / node_cpu_limit **Note** pod_cpu_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_cpu_utilization	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName, FullPodName	pod 所使用的 CPU 单元的百分比。 公式：pod_cpu_usage_total / node_cpu_limit **Note** pod_cpu_usage_total 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_cpu_utilization_over_pod_limit	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName, FullPodName	相对于容器组（pod）限制的容器组（pod）所使用的 CPU 单元的百分比。 公式：pod_cpu_usage_total / pod_cpu_limit **Note** pod_cpu_usage_total 和 pod_cpu_limit 不是直接作为指标报告，而是性能日志事件中的字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_memory_reserved_capacity	PodName、Namespace、ClusterName ClusterName	ClusterName ，Namespace ，PodName, FullPodName ClusterName ，Namespace ，Service	为 pod 预留的内存的百分比。 公式：pod_memory_request / node_memory_limit **Note** pod_memory_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_memory_utilization	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName，FullPodName	一个或多个 pod 当前正在使用的内存百分比。 公式：pod_memory_working_set / node_memory_limit **Note** pod_memory_working_set 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_memory_utilization_over_pod_limit	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName，FullPodName	相对于容器组（pod）限制的容器组（pod）所使用的内存百分比。如果容器组（pod）中的任何容器没有定义内存限制，则不会显示该指标。 公式：pod_memory_working_set / pod_memory_limit **Note** pod_memory_working_set 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_network_rx_bytes	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName, FullPodName	pod 通过网络每秒接收的字节数。 公式：sum(pod_interface_network_rx_bytes) **Note** pod_interface_network_rx_bytes 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_network_tx_bytes	PodName、Namespace、ClusterName Namespace、ClusterName Service、Namespace、ClusterName ClusterName	ClusterName，Namespace，PodName, FullPodName	pod 通过网络每秒传输的字节数。 公式：sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_cpu_request 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName ,Namespace , PodName, FullPodName	容器组（pod）的 CPU 请求。 公式：sum(container_cpu_request) **Note** pod_cpu_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_memory_request 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName ,Namespace , PodName, FullPodName	容器组（pod）的内存请求。 公式：sum(container_memory_request) **Note** pod_memory_request 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_cpu_limit 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace ， ClusterName Namespace ， ClusterName ， Service ClusterName ，Namespace ，PodName, FullPodName	为容器组（pod）中的容器定义的 CPU 限制。如果容器组（pod）中的任何容器没有定义 CPU 限制，则不会显示此指标。 公式：sum(container_cpu_limit) **Note** pod_cpu_limit 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_memory_limit 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	为容器组（pod）中的容器定义的内存限制。如果容器组（pod）中的任何容器没有定义内存限制，则不会显示该指标。 公式：sum(container_memory_limit) **Note** pod_cpu_limit 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_status_failed 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示容器组（pod）中的所有容器都已终止，并且至少有一个容器以非零状态终止或已被系统终止。
pod_status_ready 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示容器组（pod）中的所有容器都已准备就绪，且已达到 ContainerReady 的条件。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_status_running 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示容器组（pod）中的所有容器都在运行。
pod_status_scheduled 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示容器组（pod）已被调度到某个节点。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_status_unknown 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示无法获取容器组（pod）的状态。
pod_status_pending 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示集群已接受容器组（pod），但其中一个或多个容器尚未准备就绪。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_status_succeeded 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	表示容器组（pod）中的所有容器都已成功终止并且不会重启。
pod_number_of_containers 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）规范中定义的容器数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_number_of_running_containers 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中当前处于 Running 状态的容器数量。
pod_container_status_terminated 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中处于 Terminated 状态的容器数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_container_status_running 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中处于 Running 状态的容器数量。
pod_container_status_waiting 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中处于 Waiting 状态的容器数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_container_status_waiting_reason_crash_loop_back_off 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中因 CrashLoopBackOff 错误而处于待处理状态的容器数量，该错误会导致容器反复启动失败。
pod_container_status_waiting_reason_create_container_config_error 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中因 CreateContainerConfigError 而处于待处理状态的容器数量。这是因为创建容器配置时出错。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_container_status_waiting_reason_create_container_error 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中由于创建容器时出错而因 CreateContainerError 处于待处理状态的容器数量。
pod_container_status_waiting_reason_image_pull_error 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中因 ErrImagePull 、ImagePullBackOff 或 InvalidImageName 而处于待处理状态的容器数量。这些情况是由于拉取容器映像时出错造成的。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_container_status_waiting_reason_oom_killer 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中因内存不足（OOM 终止）而处于 Terminated 状态的容器数量。
pod_container_status_waiting_reason_start_error 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	报告容器组（pod）中由于启动容器时出错而因 StartError 处于待处理状态的容器数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
pod_interface_network_rx_dropped 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	容器组（pod）的网络接口接收并随后丢弃的数据包数量。
pod_interface_network_tx_dropped 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	本应传输但为容器组（pod）丢弃的数据包数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
container_cpu_utilization 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	容器所使用的 CPU 单元的百分比。 公式：container_cpu_usage_total / node_cpu_limit **Note** container_cpu_utilization 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
container_cpu_utilization_over_container_limit 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	相对于容器限制，容器所使用的 CPU 单元的百分比。如果容器没有定义 CPU 限制，则不会显示此指标。 公式：container_cpu_usage_total / container_cpu_limit **Note** container_cpu_utilization_over_container_limit 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
container_memory_utilization 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	容器所使用的内存单元的百分比。 公式：container_memory_working_set / node_memory_limit **Note** container_memory_utilization 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
container_memory_utilization_over_container_limit 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	相对于容器限制，容器所使用的内存单元的百分比。如果容器没有定义内存限制，则不会显示此指标。 公式：container_memory_working_set / container_memory_limit **Note** container_memory_utilization_over_container_limit 不是直接作为指标报告，而是性能日志事件中的一个字段。有关更多信息，请参阅 Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
container_memory_failures_total 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights。其在 Windows 上不可用。		ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	容器遇到的内存分配失败的次数。
pod_number_of_container_restarts	PodName、Namespace 、ClusterName		一个 pod 中容器重新启动的总次数。
service_number_of_running_pods	Service、Namespace 、ClusterName ClusterName		运行集群中的服务的 pod 的数量。
replicas_desired 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName	工作负载规范中定义的工作负载所需的容器组（pod）数量。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
replicas_ready 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName	已达到就绪状态的工作负载的容器组（pod）数量。
status_replicas_available 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName	工作负载可用的容器组（pod）数量。当容器组（pod）已就绪工作负载规范中定义的 minReadySeconds 时，容器组（pod）才可用。
status_replicas_unavailable 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName PodName, Namespace , ClusterName	工作负载不可用的容器组（pod）数量。当容器组（pod）已就绪工作负载规范中定义的 minReadySeconds 时，容器组（pod）才可用。如果容器组（pod）不符合此标准，则它们不可用。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述

`apiserver_storage_objects`
 	
此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights
		`ClusterName`
		
		`ClusterName`，`resource`
			上次检查时存储在 etcd 中的对象数量。

`apiserver_request_total`
 	
此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights
		`ClusterName`
		
		`ClusterName`，`code`, `verb`
			向 Kubernetes API 服务器发出的 API 请求总数。

`apiserver_request_duration_seconds`
 	
此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights
		`ClusterName`
		
		`ClusterName`，`verb`
			向 Kubernetes API 服务器发出的 API 请求的响应延迟。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
apiserver_admission_controller_admission_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，operation	准入控制器延迟（以秒为单位）。准入控制器是拦截向 Kubernetes API 服务器发出的请求的代码。
rest_client_request_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，operation	客户端在调用 Kubernetes API 服务器时遇到的响应延迟。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。
rest_client_requests_total 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，code, method	客户端向 Kubernetes API 服务器发出的 API 请求总数。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
etcd_request_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，operation	对 Etcd 的 API 调用的响应延迟。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。
apiserver_storage_size_bytes 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，endpoint	物理分配的存储数据库文件的大小（以字节为单位）。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。
apiserver_longrunning_requests 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，resource	向 Kubernetes API 服务器发出的长时间运行的活跃请求数。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
apiserver_current_inflight_requests 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，request_kind	Kubernetes API 服务器正在处理的请求数。
apiserver_admission_webhook_admission_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，name	准入 Webhook 延迟（以秒为单位）。准入 Webhook 是 HTTP 回调，用于接收准入请求并对其进行一些处理。
apiserver_admission_step_admission_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，operation	准入子步骤延迟（以秒为单位）。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述
apiserver_requested_deprecated_apis 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，group	向 Kubernetes API 服务器上已弃用的 API 发出的请求数。
apiserver_request_total_5XX 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，code, verb	向 Kubernetes API 服务器发出的请求数，这些请求以 5XX HTTP 响应代码为响应。
apiserver_storage_list_duration_seconds 此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights		ClusterName ClusterName，resource	列出 Etcd 中的对象的响应延迟。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。

指标名称	任何版本的 Container Insights 的维度	适用于针对 Amazon EKS 增强了可观测性的 Container Insights 的其他维度	描述

`apiserver_current_inqueue_requests`
 	
此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights
		`ClusterName`
		
		`ClusterName`，`request_kind`
			Kubernetes API 服务器排队的请求数。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。

`apiserver_flowcontrol_rejected_requests_total`
 	
此指标仅适用于针对 Amazon EKS 增强了可观测性的 Container Insights
		`ClusterName`
		
		`ClusterName`，`reason`
			API 优先级和公平性子系统拒绝的请求数。此指标是实验性的，在将来的 Kubernetes 版本中可能会发生变化。

NVIDIA GPU 指标

从 CloudWatch 代理版本 `1.300034.0` 开始，针对 Amazon EKS 增强了可观测性的 Container Insights 默认从 EKS 工作负载收集 NVIDIA GPU 指标。必须使用 CloudWatch Observability EKS 附加组件版本 `v1.3.0-eksbuild.1` 或更高版本安装 CloudWatch 代理。有关更多信息，请参阅 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理。本节的表中列出了这些会被收集的 NVIDIA GPU 指标。

要让 Container Insights 收集 NVIDIA GPU 指标，必须满足以下先决条件：

- 必须将针对 Amazon EKS 增强了可观测性的 Container Insights 与 Amazon CloudWatch Observability EKS 附加组件版本 `v1.3.0-eksbuild.1` 或更高版本结合使用。

- 集群中必须安装适用于 Kubernetes 的 NVIDIA 设备插件。
- 集群的节点上必须安装 NVIDIA 容器工具包。例如，使用必要的组件构建 Amazon EKS 优化版加速型 AMI。

您可以将起始 CloudWatch 代理配置文件中的 `accelerated_compute_metrics` 选项设置为 `false`，从而选择不收集 NVIDIA GPU 指标。有关更多信息和选择不收集配置的示例，请参阅（可选）其他配置。

指标名称	Dimensions	描述
container_gpu_memory_total	ClusterName ClusterName，Namespace，PodName，ContainerName ClusterName，Namespace，PodName，FullPodName，ContainerName ClusterName，Namespace，PodName，FullPodName，ContainerName，GpuDevice	分配给容器的 GPU 上的帧缓冲区总大小（以字节为单位）。
container_gpu_memory_used	ClusterName ClusterName，Namespace，PodName，ContainerName ClusterName，Namespace，PodName，FullPodName，ContainerName ClusterName，Namespace，PodName，FullPodName，ContainerName，GpuDevice	分配给容器的 GPU 上使用的帧缓冲区字节数。
container_gpu_memo	ClusterName	分配给容器的 GPU 已使用的帧缓冲区百分比。

指标名称	Dimensions	描述
ry_utiliz ation	ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	
container _gpu_powe r_draw	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	分配给容器的 GPU 的功耗（以瓦特为单位）。
container _gpu_temp erature	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	分配给容器的 GPU 的温度（以摄氏度为单位）。

指标名称	Dimensions	描述
container_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	分配给容器的 GPU 的利用率百分比。
node_gpu_memory_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	分配给节点的 GPU 上的帧缓冲区总大小（以字节为单位）。
node_gpu_memory_used	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	分配给节点的 GPU 上使用的帧缓冲区字节数。

指标名称	Dimensions	描述
node_gpu_memory_utilization	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，GpuDevice	分配给节点的 GPU 上使用的帧缓冲区百分比。
node_gpu_power_draw	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，GpuDevice	分配给节点的 GPU 的功耗（以瓦特为单位）。
node_gpu_temperature	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，GpuDevice	分配给节点的 GPU 的温度（以摄氏度为单位）。
node_gpu_utilization	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，GpuDevice	分配给节点的 GPU 的利用率百分比。

指标名称	Dimensions	描述
pod_gpu_memory_total	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	分配给 Pod 的 GPU 上的帧缓冲区总大小（以字节为单位）。
pod_gpu_memory_used	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	分配给 Pod 的 GPU 上使用的帧缓冲区字节数。

指标名称	Dimensions	描述
pod_gpu_memory_utilization	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	分配给 Pod 的 GPU 上使用的帧缓冲区百分比。
pod_gpu_power_draw	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	分配给 Pod 的 GPU 的功耗（以瓦特为单位）。

指标名称	Dimensions	描述
pod_gpu_temperature	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	分配给 Pod 的 GPU 的温度（以摄氏度为单位）。
pod_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	分配给 Pod 的 GPU 的利用率百分比。

AWS Trainium 和 AWS Inferentia 的 AWS Neuron 指标

从 CloudWatch 代理版本 1.300036.0 开始，针对 Amazon EKS 增强了可观测性的 Container Insights 默认从 AWS Trainium 和 AWS Inferentia 加速器收集加速计算指标。必须使用 CloudWatch Observability EKS 附加组件版本 v1.5.0-eksbuild.1 或更高版本安装 CloudWatch 代理。有关附加组件的更多信息，请参阅 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装

CloudWatch 代理。有关 AWS Trainium 的更多信息，请参阅 AWS Trainium。有关 AWS Inferentia 的更多信息，请参阅 AWS Inferentia。

要让 Container Insights 收集 AWS Neuron 指标，必须满足以下先决条件：

- 必须将针对 Amazon EKS 增强了可观测性的 Container Insights 与 Amazon CloudWatch Observability EKS 附加组件版本 `v1.5.0-eksbuild.1` 或更高版本结合使用。
- Neuron 驱动程序必须安装在集群的节点上。
- Neuron 设备插件必须安装在集群上。例如，使用必要的组件构建 Amazon EKS 优化版加速型 AMI。

本节的表中列出了将收集的指标。这些指标是为 AWS Trainium、AWS Inferentia 和 AWS Inferentia2 收集的。

CloudWatch 代理从 Neuron Monitor 收集这些指标，并进行必要的 Kubernetes 资源关联，以在容器组（pod）和容器级别提供指标

指标名称	Dimensions	描述

`container_neuroncore_utilization`	`ClusterName`
	
	`ClusterName`，`Namespace`，`PodName`, `ContainerName`
	
	`ClusterName`，`Namespace`，`PodName`, `FullPodName`，`ContainerName`
	
	`ClusterName`，`Namespace`，`PodName`, `FullPodName`，`ContainerName`，`NeuronDevice`，`NeuronCore`
		分配给容器的 NeuronCore 在捕获期内的 NeuronCore 利用率。
		
		单位：百分比

`container_neuroncore_memory_usage_constants`	`ClusterName`
	
	`ClusterName`，`Namespace`，`PodName`, `ContainerName`
		分配给容器的 NeuronCore 在训练期间用于常量的设备内存量（或推理期间的权重）。
		
		单位：字节

指标名称	Dimensions	描述
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	
container_neuroncore_memory_usage_model_code	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	分配给容器的 NeuronCore 用于模型可执行代码的设备内存量。 单位：字节
container_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	分配给容器的 NeuronCore 用于模型共享暂存器的设备内存量。此内存区域保留用于模型。 单位：字节

指标名称	Dimensions	描述
container_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	分配给容器的 NeuronCore 用于 Neuron 运行时的设备内存量。 单位：字节
container_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	分配给容器的 NeuronCore 用于张量的设备内存量。 单位：字节

指标名称	Dimensions	描述
container_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	分配给容器的 NeuronCore 使用的内存总量。 单位：字节
container_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice	节点上 Neuron 设备的片上 SRAM 和设备内存的已校正和未校正 ECC 事件数。 单位：计数

指标名称	Dimensions	描述
pod_neuroncore_utilization	ClusterName ClusterName ，Namespace ClusterName ，Namespace ，Service ClusterName ，Namespace ，PodName, FullPodName ClusterName ，Namespace ，PodName, FullPodName ，NeuronDevice ，NeuronCore	分配给容器组（pod）的 NeuronCore 在捕获期内的 NeuronCore 利用率。 单位：百分比
pod_neuroncore_memory_usage_constants	ClusterName ClusterName ，Namespace ClusterName ，Namespace ，Service ClusterName ，Namespace ，PodName, FullPodName ClusterName ，Namespace ，PodName, FullPodName ，NeuronDevice ，NeuronCore	分配给容器组（pod）的 NeuronCore 在训练期间用于常量的设备内存量（或推理期间的权重）。 单位：字节

指标名称	Dimensions	描述
pod_neuroncore_memory_usage_model_code	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	分配给容器组（pod）的 NeuronCore 用于模型可执行代码的设备内存量。 单位：字节
pod_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	分配给容器组（pod）的 NeuronCore 用于模型共享暂存器的设备内存量。此内存区域保留用于模型。 单位：字节

指标名称	Dimensions	描述
pod_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	分配给容器组（pod）的 NeuronCore 用于 Neuron 运行时的设备内存量。 单位：字节
pod_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	分配给容器组（pod）的 NeuronCore 用于张量的设备内存量。 单位：字节

指标名称	Dimensions	描述
pod_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	分配给容器组（pod）的 NeuronCore 使用的内存总量。 单位：字节
pod_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice	分配给容器组（pod）的 Neuron 设备的片上 SRAM 和设备内存的已校正和未校正 ECC 事件数。 单位：字节

指标名称	Dimensions	描述
node_neuroncore_utilization	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 在捕获期内的 NeuronCore 利用率。 单位：百分比
node_neuroncore_memory_usage_constants	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 在训练期间用于常量的设备内存量（或推理期间的权重）。 单位：字节
node_neuroncore_memory_usage_model_code	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 用于模型可执行代码的设备内存量。 单位：字节
node_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 用于模型共享暂存器的设备内存量。这保留用于模型的内存区域。 单位：字节

指标名称	Dimensions	描述
node_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 用于 Neuron 运行时的设备内存量。 单位：字节
node_neuroncore_memory_usage_tensors	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 用于张量的设备内存量。 单位：字节
node_neuroncore_memory_usage_total	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceType ，InstanceId ，NodeName，NeuronDevice ，NeuronCore	分配给节点的 NeuronCore 使用的内存总量。 单位：字节
node_neuron_execution_errors_total	ClusterName ClusterName ，InstanceId ，NodeName	节点上执行错误的总数。这是由 CloudWatch 代理通过汇总以下类型的错误来计算的：generic、numerical 、transient 、model、runtime 和 hardware 单位：计数

指标名称	Dimensions	描述
node_neuron_device_runtime_memory_used_bytes	ClusterName ClusterName，InstanceId，NodeName	节点上 Neuron 设备内存使用总量（以字节为单位）。 单位：字节
node_neuron_execution_latency	ClusterName ClusterName，InstanceId，NodeName	Neuron 运行时测量的节点上执行的延迟（以秒为单位）。 单位：秒
node_neuron_device_hw_ecc_events_total	ClusterName ClusterName，InstanceId，NodeName ClusterName，InstanceId，NodeName, NeuronDevice	节点上 Neuron 设备的片上 SRAM 和设备内存的已校正和未校正 ECC 事件数。 单位：计数

AWS Elastic Fabric Adapter（EFA）指标

从 CloudWatch 代理版本 `1.300037.0` 开始，针对 Amazon EKS 增强了可观测性的 Container Insights 从 Linux 实例上的 Amazon EKS 集群收集 AWS Elastic Fabric Adapter（EFA）指标。必须使用 CloudWatch Observability EKS 附加组件版本 `v1.5.2-eksbuild.1` 或更高版本安装 CloudWatch 代理。有关附加组件的更多信息，请参阅 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理。有关 AWS Elastic Fabric Adapter（EFA）的更多信息，请参阅 Elastic Fabric Adapter。

要让 Container Insights 收集 AWS Elastic Fabric Adapter 指标，必须满足以下先决条件：

- 必须将针对 Amazon EKS 增强了可观测性的 Container Insights 与 Amazon CloudWatch Observability EKS 附加组件版本 `v1.5.2-eksbuild.1` 或更高版本结合使用。
- EFA 设备插件必须安装在集群上。有关更多信息，请参阅 GitHub 上的 aws-efa-k8s-device-plugin。

下表中列出了收集的指标。

指标名称	Dimensions	描述
container_efa_rx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	分配给容器的 EFA 设备每秒接收的字节数。 单位：字节/秒
container_efa_tx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	分配给容器的 EFA 设备每秒传输的字节数。 单位：字节/秒
container_efa_rx_dropped	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName	分配给容器的 EFA 设备接收然后丢弃的数据包数量。 单位：计数/秒

指标名称	Dimensions	描述
	ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	
container_efa_rdma_read_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	分配给容器的 EFA 设备使用远程直接内存访问读取操作每秒接收的字节数。 单位：字节/秒
container_efa_rdma_write_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	分配给容器的 EFA 设备使用远程直接内存访问读取操作每秒传输的字节数。 单位：字节/秒

指标名称	Dimensions	描述
container_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	分配给容器的 EFA 设备在远程直接内存访问写入操作期间每秒接收的字节数。 单位：字节/秒
pod_efa_rx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备每秒接收的字节数。 单位：字节/秒

指标名称	Dimensions	描述
pod_efa_tx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备每秒传输的字节数。 单位：字节/秒
pod_efa_rx_dropped	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备接收然后丢弃的数据包数量。 单位：计数/秒

指标名称	Dimensions	描述
pod_efa_rdma_read_bytes	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备使用远程直接内存访问读取操作每秒接收的字节数。 单位：字节/秒
pod_efa_rdma_write_bytes	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备使用远程直接内存访问读取操作每秒传输的字节数。 单位：字节/秒

指标名称	Dimensions	描述
pod_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	分配给容器组（pod）的 EFA 设备在远程直接内存访问写入操作期间每秒接收的字节数。 单位：字节/秒
node_efa_rx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	分配给节点的 EFA 设备每秒接收的字节数。 单位：字节/秒
node_efa_tx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	分配给节点的 EFA 设备每秒传输的字节数。 单位：字节/秒

指标名称	Dimensions	描述
node_efa_rx_dropped	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，EfaDevice	分配给节点的 EFA 设备接收然后丢弃的数据包数量。 单位：计数/秒
node_efa_rdma_read_bytes	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，EfaDevice	分配给节点的 EFA 设备使用远程直接内存访问读取操作每秒接收的字节数。 单位：字节/秒
pod_efa_rdma_write_bytes	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，EfaDevice	分配给容器组（pod）的 EFA 设备使用远程直接内存访问读取操作每秒传输的字节数。 单位：字节/秒
node_efa_rdma_write_recv_bytes	ClusterName ClusterName ，InstanceId ，NodeName ClusterName ，InstanceId ，InstanceType ，NodeName，EfaDevice	分配给节点的 EFA 设备在远程直接内存访问写入操作期间每秒接收的字节数。 单位：字节/秒

Container Insights 性能日志参考

本节包含有关 Container Insights 如何使用性能日志事件来收集指标的参考信息。当您部署 Container Insights 时，它会自动为性能日志事件创建日志组。您无需自行创建该日志组。

主题

- Amazon ECS 的 Container Insights 性能日志事件
- Amazon EKS 和 Kubernetes 的 Container Insights 性能日志事件
- Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段

Amazon ECS 的 Container Insights 性能日志事件

以下是 Container Insights 从 Amazon ECS 收集的性能日志事件的实例。

这些日志在 CloudWatch Logs 中位于名为 `/aws/ecs/containerinsights/`*`CLUSTER_NAME`*`/performance` 的日志组中。在该日志组中，每个容器实例都将有一个名为 `AgentTelemetry-`*`CONTAINER_INSTANCE_ID`* 的日志流。

您可以使用查询来查询这些日志，例如使用 `{ $.Type = "Container" }` 来查看所有容器日志事件。

类型：容器

```
{
 "Version":"0",
 "Type":"Container",
 "ContainerName":"sleep",
 "TaskId":"7ac4dfba69214411b4783a3b8189c9ba",
 "TaskDefinitionFamily":"sleep360",
 "TaskDefinitionRevision":"1",
 "ContainerInstanceId":"0d7650e6dec34c1a9200f72098071e8f",
 "EC2InstanceId":"i-0c470579dbcdbd2f3",
 "ClusterName":"MyCluster",
 "Image":"busybox",
 "ContainerKnownStatus":"RUNNING",
 "Timestamp":1623963900000,
 "CpuUtilized":0.0,
 "CpuReserved":10.0,
 "MemoryUtilized":0,
 "MemoryReserved":10,
```

```
 "StorageReadBytes":0,
 "StorageWriteBytes":0,
 "NetworkRxBytes":0,
 "NetworkRxDropped":0,
 "NetworkRxErrors":0,
 "NetworkRxPackets":14,
 "NetworkTxBytes":0,
 "NetworkTxDropped":0,
 "NetworkTxErrors":0,
 "NetworkTxPackets":0
}
```

类型：任务

```
{
    "Version": "0",
    "Type": "Task",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "TaskDefinitionFamily": "sleep360",
    "TaskDefinitionRevision": "1",
    "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
    "EC2InstanceId": "i-0c470579dbcdbd2f3",
    "ClusterName": "MyCluster",
    "AccountID": "637146863587",
    "Region": "us-west-2",
    "AvailabilityZone": "us-west-2b",
    "KnownStatus": "RUNNING",
    "LaunchType": "EC2",
    "PullStartedAt": 1623963608201,
    "PullStoppedAt": 1623963610065,
    "CreatedAt": 1623963607094,
    "StartedAt": 1623963610382,
    "Timestamp": 1623963900000,
    "CpuUtilized": 0.0,
    "CpuReserved": 10.0,
    "MemoryUtilized": 0,
    "MemoryReserved": 10,
    "StorageReadBytes": 0,
    "StorageWriteBytes": 0,
    "NetworkRxBytes": 0,
    "NetworkRxDropped": 0,
    "NetworkRxErrors": 0,
    "NetworkRxPackets": 14,
```

```
    "NetworkTxBytes": 0,
    "NetworkTxDropped": 0,
    "NetworkTxErrors": 0,
    "NetworkTxPackets": 0,
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "CpuUtilized",
                    "Unit": "None"
                },
                {
                    "Name": "CpuReserved",
                    "Unit": "None"
                },
                {
                    "Name": "MemoryUtilized",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "MemoryReserved",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "StorageReadBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "StorageWriteBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkRxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkTxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "EBSFilesystemSize",
```

```
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "ClusterName",
                    "TaskDefinitionFamily"
                ]
            ]
        }
    ]
}
```

类型：服务

```
{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "RunningTaskCount",
                    "Unit": "Count"
                },
```

```
                {
                    "Name": "PendingTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "DeploymentCount",
                    "Unit": "Count"
                },
                {
                    "Name": "TaskSetCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

类型：卷

```
{
    "Version": "0",
    "Type": "Volume",
    "TaskDefinitionFamily": "myCITaskDef",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "ClusterName": "myCICluster",
    "ServiceName": "myCIService",
    "VolumeId": "vol-1233436545ff708cb",
    "InstanceId": "i-0c470579dbcdbd2f3",
    "LaunchType": "EC2",
    "VolumeName": "MyVolumeName",
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
```

```
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "VolumeName",
                    "TaskDefinitionFamily",
                    "ClusterName"
                ],
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

类型：集群

```
{
    "Version": "0",
    "Type": "Cluster",
    "ClusterName": "myCICluster",
    "Timestamp": 1561587300000,
    "TaskCount": 5,
    "ContainerInstanceCount": 5,
    "ServiceCount": 2,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "TaskCount",
                    "Unit": "Count"
                },
                {
```

```
                    "Name": "ContainerInstanceCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ServiceCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Amazon EKS 和 Kubernetes 的 Container Insights 性能日志事件

以下是 Container Insights 从 Amazon EKS 和 Kubernetes 集群中收集的性能日志事件的示例。

类型：Node

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
```

```
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
```

```
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        },
        {
          "Name": "node_cpu_usage_total"
        },
        {
          "Name": "node_cpu_limit"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_working_set"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_limit"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "/proc",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567096682364",
  "Type": "Node",
```

```
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_cpu_limit": 4000,
  "node_cpu_request": 1130,
  "node_cpu_reserved_capacity": 28.249999999999996,
  "node_cpu_usage_system": 33.794636630852764,
  "node_cpu_usage_total": 136.47852169244098,
  "node_cpu_usage_user": 71.67075111567326,
  "node_cpu_utilization": 3.4119630423110245,
  "node_memory_cache": 3103297536,
  "node_memory_failcnt": 0,
  "node_memory_hierarchical_pgfault": 0,
  "node_memory_hierarchical_pgmajfault": 0,
  "node_memory_limit": 16624865280,
  "node_memory_mapped_file": 406646784,
  "node_memory_max_usage": 4230746112,
  "node_memory_pgfault": 0,
  "node_memory_pgmajfault": 0,
  "node_memory_request": 1115684864,
  "node_memory_reserved_capacity": 6.7109407818311055,
  "node_memory_rss": 798146560,
  "node_memory_swap": 0,
  "node_memory_usage": 3901444096,
  "node_memory_utilization": 6.601302600149552,
  "node_memory_working_set": 1097457664,
  "node_network_rx_bytes": 35918.392817386324,
  "node_network_rx_dropped": 0,
  "node_network_rx_errors": 0,
  "node_network_rx_packets": 157.67565245448117,
  "node_network_total_bytes": 68264.20276554905,
  "node_network_tx_bytes": 32345.80994816272,
  "node_network_tx_dropped": 0,
  "node_network_tx_errors": 0,
  "node_network_tx_packets": 154.21455923431654,
  "node_number_of_running_containers": 16,
  "node_number_of_running_pods": 13
}
```


类型：NodeFS


```
{
```

```
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_filesystem_utilization"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097939726",
  "Type": "NodeFS",
  "Version": "0",
  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_filesystem_available": 17298395136,
  "node_filesystem_capacity": 21462233088,
  "node_filesystem_inodes": 10484720,
  "node_filesystem_inodes_free": 10367158,
```

```
  "node_filesystem_usage": 4163837952,
  "node_filesystem_utilization": 19.400767547940255
}
```

类型：NodeDiskIO

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeDiskIO",
  "Version": "0",
  "device": "/dev/nvme0n1",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_diskio_io_service_bytes_async": 9750.505814277016,
  "node_diskio_io_service_bytes_read": 0,
  "node_diskio_io_service_bytes_sync": 230.6174506688036,
  "node_diskio_io_service_bytes_total": 9981.123264945818,
  "node_diskio_io_service_bytes_write": 9981.123264945818,
  "node_diskio_io_serviced_async": 1.153087253344018,
  "node_diskio_io_serviced_read": 0,
  "node_diskio_io_serviced_sync": 0.03603397666700056,
  "node_diskio_io_serviced_total": 1.1891212300110185,
  "node_diskio_io_serviced_write": 1.1891212300110185
}
```

类型：NodeNet

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
```

```
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeNet",
  "Version": "0",
  "interface": "eni972f6bfa9a0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_interface_network_rx_bytes": 3163.008420864309,
  "node_interface_network_rx_dropped": 0,
  "node_interface_network_rx_errors": 0,
  "node_interface_network_rx_packets": 16.575629266820258,
  "node_interface_network_total_bytes": 3518.3935157426017,
  "node_interface_network_tx_bytes": 355.385094878293,
  "node_interface_network_tx_dropped": 0,
  "node_interface_network_tx_errors": 0,
  "node_interface_network_tx_packets": 3.9997714100370625
}
```

类型：Pod

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_rx_bytes"
```

```
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_tx_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization_over_pod_limit"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization_over_pod_limit"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_reserved_capacity"
```

```
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "pod_number_of_container_restarts"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097351092",
```

```
  "Type": "Pod",
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_cpu_limit": 200,
  "pod_cpu_request": 200,
  "pod_cpu_reserved_capacity": 5,
  "pod_cpu_usage_system": 1.4504841104992765,
  "pod_cpu_usage_total": 5.817016867430125,
  "pod_cpu_usage_user": 1.1281543081661038,
  "pod_cpu_utilization": 0.14542542168575312,
  "pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
  "pod_memory_cache": 8192,
  "pod_memory_failcnt": 0,
  "pod_memory_hierarchical_pgfault": 0,
  "pod_memory_hierarchical_pgmajfault": 0,
  "pod_memory_limit": 104857600,
  "pod_memory_mapped_file": 0,
  "pod_memory_max_usage": 25268224,
  "pod_memory_pgfault": 0,
  "pod_memory_pgmajfault": 0,
  "pod_memory_request": 104857600,
  "pod_memory_reserved_capacity": 0.6307275170893897,
  "pod_memory_rss": 22777856,
  "pod_memory_swap": 0,
  "pod_memory_usage": 25141248,
  "pod_memory_utilization": 0.10988455961791709,
  "pod_memory_utilization_over_pod_limit": 17.421875,
  "pod_memory_working_set": 18268160,
```

```
  "pod_network_rx_bytes": 9880.697124714186,
  "pod_network_rx_dropped": 0,
  "pod_network_rx_errors": 0,
  "pod_network_rx_packets": 107.80005532263283,
  "pod_network_total_bytes": 10158.829201483635,
  "pod_network_tx_bytes": 278.13207676944796,
  "pod_network_tx_dropped": 0,
  "pod_network_tx_errors": 0,
  "pod_network_tx_packets": 1.146027574644318,
  "pod_number_of_container_restarts": 0,
  "pod_number_of_containers": 1,
  "pod_number_of_running_containers": 1,
  "pod_status": "Running"
}
```

类型：PodNet

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "PodNet",
  "Version": "0",
  "interface": "eth0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
```

```
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_interface_network_rx_bytes": 9880.697124714186,
  "pod_interface_network_rx_dropped": 0,
  "pod_interface_network_rx_errors": 0,
  "pod_interface_network_rx_packets": 107.80005532263283,
  "pod_interface_network_total_bytes": 10158.829201483635,
  "pod_interface_network_tx_bytes": 278.13207676944796,
  "pod_interface_network_tx_dropped": 0,
  "pod_interface_network_tx_errors": 0,
  "pod_interface_network_tx_packets": 1.146027574644318
}
```

类型：容器

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-
sample",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "Container",
  "Version": "0",
  "container_cpu_limit": 200,
  "container_cpu_request": 200,
  "container_cpu_usage_system": 1.87958283771964,
```

```
  "container_cpu_usage_total": 6.159993652997942,
  "container_cpu_usage_user": 1.6707403001952357,
  "container_cpu_utilization": 0.15399984132494854,
  "container_memory_cache": 8192,
  "container_memory_failcnt": 0,
  "container_memory_hierarchical_pgfault": 0,
  "container_memory_hierarchical_pgmajfault": 0,
  "container_memory_limit": 104857600,
  "container_memory_mapped_file": 0,
  "container_memory_max_usage": 24580096,
  "container_memory_pgfault": 0,
  "container_memory_pgmajfault": 0,
  "container_memory_request": 104857600,
  "container_memory_rss": 22736896,
  "container_memory_swap": 0,
  "container_memory_usage": 24453120,
  "container_memory_utilization": 0.10574541028701798,
  "container_memory_working_set": 17580032,
  "container_status": "Running",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "number_of_container_restarts": 0
}
```

类型：ContainerFS


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "ContainerFS",
  "Version": "0",

  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
```

```
  }
}
```

类型：集群

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
}
```

类型：“ClusterService”

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
```

```
        {
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
  },
  "service_number_of_running_pods": 1
}
```

类型：ClusterNamespace

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "namespace_number_of_running_pods"
```

```
        }
      ],
      "Dimensions": [
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097594160",
  "Type": "ClusterNamespace",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch"
  },
  "namespace_number_of_running_pods": 7
}
```

Amazon EKS 和 Kubernetes 的性能日志事件中的相关字段

对于 Amazon EKS 和 Kubernetes，容器化的 CloudWatch 代理将数据作为性能日志事件发出。这使 CloudWatch 能够摄取和存储高基数数据。CloudWatch 使用性能日志事件中的数据在集群、节点和 pod 级别创建聚合的 CloudWatch 指标，而不会丢失详细信息。

下表列出了这些性能日志事件中与 Container Insights 指标数据收集相关的字段。您可以使用 CloudWatch Logs Insights 查询其中的任何字段以收集数据或调查问题。有关更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。

类型	日志字段	来源	公式或备注
Pod	pod_cpu_utilization	计算	公式：pod_cpu_u

类型	日志字段		来源	公式或备注
				sage_total / node_cpu_limit
Pod	pod_cpu_usage_total pod_cpu_usage_total 心为单位报道。	以千分之一核	cadvisor	
Pod	pod_cpu_limit		计算	公式：sum(container_cpu_limit) sum(container_cpu_limit) 包括已经完成的容器组（pod）。 如果 pod 中的任何容器未定义 CPU 限制，则该字段不会显示在日志事件中。这包括 init 容器。

类型	日志字段	来源	公式或备注
Pod	pod_cpu_request	计算	公式：sum(container_cpu_request) 不保证设置 container_cpu_request 。仅在总和中包括设置的字段。
Pod	pod_cpu_utilization_over_pod_limit	计算	公式：pod_cpu_usage_total / pod_cpu_limit
Pod	pod_cpu_reserved_capacity	计算	公式：pod_cpu_request / node_cpu_limit
Pod	pod_memory_utilization	计算	公式：pod_memory_working_set / node_memory_limit 它是 pod 内存使用量相比节点内存限制的百分比。

类型	日志字段	来源	公式或备注
Pod	pod_memory_working_set	cadvisor	
Pod	pod_memory_limit	计算	公式：sum(container_memory_limit) 如果 pod 中的任何容器没有定义内存限制，则该字段不会显示在日志事件中。这包括 init 容器。
Pod	pod_memory_request	计算	公式：sum(container_memory_request) 不保证设置 container_memory_request 。仅在总和中包括设置的字段。

类型	日志字段	来源	公式或备注
Pod	pod_memory_utilization_over_pod_limit	计算	公式：pod_memory_working_set / pod_memory_limit 如果 pod 中的任何容器没有定义内存限制，则该字段不会显示在日志事件中。这包括 init 容器。
Pod	pod_memory_reserved_capacity	计算	公式：pod_memory_request / node_memory_limit
Pod	pod_network_tx_bytes	计算	公式：sum(pod_interface_network_tx_bytes) 该数据适用于每个 pod 的所有网络接口。CloudWatch 代理计算总数并添加指标提取规则。

类型	日志字段	来源	公式或备注
Pod	pod_network_rx_bytes	计算	公式：sum(pod_interface_network_rx_bytes)
Pod	pod_network_total_bytes	计算	公式：pod_network_rx_bytes + pod_network_tx_bytes
PodNet	pod_interface_network_rx_bytes	cadvisor	该数据是 pod 网络接口每秒接收的网络字节数。
PodNet	pod_interface_network_tx_bytes	cadvisor	该数据是 pod 网络接口每秒发送的网络字节数。
容器	container_cpu_usage_total	cadvisor	
容器	container_cpu_limit	cadvisor	不保证设置该字段。如果未设置，则不会发出该指标。
容器	container_cpu_request	cadvisor	不保证设置该字段。如果未设置，则不会发出该指标。

类型	日志字段	来源	公式或备注
容器	container_memory_working_set	cadvisor	
容器	container_memory_limit	容器组（pod）	不保证设置该字段。如果未设置，则不会发出该指标。
容器	container_memory_request	容器组（pod）	不保证设置该字段。如果未设置，则不会发出该指标。
节点	node_cpu_utilization	计算	公式：node_cpu_usage_total / node_cpu_limit
节点	node_cpu_usage_total	cadvisor	
节点	node_cpu_limit	/proc	

类型	日志字段	来源	公式或备注
节点	node_cpu_request	计算	公式：sum(pod_cpu_request) 对于 cronjobs，node_cpu_request 还包括来自已完成的容器组（pod）的请求。这可能会导致 node_cpu_reserved_capacity 的值很高。
节点	node_cpu_reserved_capacity	计算	公式：node_cpu_request / node_cpu_limit
节点	node_memory_utilization	计算	公式：node_memory_working_set / node_memory_limit
节点	node_memory_working_set	cadvisor	
节点	node_memory_limit	/proc	

类型	日志字段	来源	公式或备注
节点	node_memory_request	计算	公式：sum(pod_memory_request)
节点	node_memory_reserved_capacity	计算	公式：node_memory_request / node_memory_limit
节点	node_network_rx_bytes	计算	公式：sum(node_interface_network_rx_bytes)
节点	node_network_tx_bytes	计算	公式：sum(node_interface_network_tx_bytes)
节点	node_network_total_bytes	计算	公式：node_network_rx_bytes + node_network_tx_bytes
节点	node_number_of_running_pods	Pod 列表	
节点	node_number_of_running_containers	Pod 列表	

类型	日志字段	来源	公式或备注
NodeNet	node_interface_network_rx_bytes	cadvisor	该数据是工作线程节点网络接口每秒接收的网络字节数。
NodeNet	node_interface_network_tx_bytes	cadvisor	该数据是工作线程节点网络接口每秒发送的网络字节数。
NodeFS	node_filesystem_capacity	cadvisor	
NodeFS	node_filesystem_usage	cadvisor	
NodeFS	node_filesystem_utilization	计算	公式：node_filesystem_usage / node_filesystem_capacity 该数据适用于每个设备名称。
集群	cluster_failed_node_count	API 服务器	
集群	cluster_node_count	API 服务器	
服务	service_number_of_running_pods	API 服务器	
Namespace	namespace_number_of_running_pods	API 服务器	

指标计算示例

本节包含的示例说明了如何计算上表中的某些值。

假设您具有一个处于以下状态的集群。

```
Node1
   node_cpu_limit = 4
   node_cpu_usage_total = 3

   Pod1
     pod_cpu_usage_total = 2

     Container1
        container_cpu_limit = 1
        container_cpu_request = 1
        container_cpu_usage_total = 0.8

     Container2
        container_cpu_limit = null
        container_cpu_request = null
        container_cpu_usage_total = 1.2

   Pod2
     pod_cpu_usage_total = 0.4

     Container3
        container_cpu_limit = 1
        container_cpu_request = 0.5
        container_cpu_usage_total = 0.4

Node2
   node_cpu_limit = 8
   node_cpu_usage_total = 1.5

   Pod3
     pod_cpu_usage_total = 1

     Container4
        container_cpu_limit = 2
        container_cpu_request = 2
        container_cpu_usage_total = 1
```

下表说明了如何使用该数据计算 pod CPU 指标。

指标	公式	Pod1	Pod2	Pod3
pod_cpu_utilization	pod_cpu_usage_total / node_cpu_limit	2 / 4 = 50%	0.4 / 4 = 10%	1 / 8 = 12.5%
pod_cpu_utilization_over_pod_limit	pod_cpu_usage_total / sum(container_cpu_limit)	不适用，因为没有定义 Container 2 的 CPU 限制	0.4 / 1 = 40%	1 / 2 = 50%
pod_cpu_reserved_capacity	sum(container_cpu_request) / node_cpu_limit	(1 + 0) / 4 = 25%	0.5 / 4 = 12.5%	2 / 8 = 25%

下表说明了如何使用该数据计算节点 CPU 指标。

指标	公式	Node1	Node2
node_cpu_utilization	node_cpu_usage_total / node_cpu_limit	3 / 4 = 75%	1.5 / 8 = 18.75%
node_cpu_reserved_capacity	sum(pod_cpu_request) / node_cpu_limit	1.5 / 4 = 37.5%	2 / 8 = 25%

Container Insights Prometheus 指标监控

适用于 Prometheus 的 CloudWatch Container Insights 监控可以自动发现来自容器化系统和工作负载的 Prometheus 指标。Prometheus 是一个开源系统监控和警报工具包。有关更多信息，请参阅 Prometheus 文档中的什么是 Prometheus？

Amazon Elastic Container Service、Amazon Elastic Kubernetes Service 和在 Amazon EC2 实例上运行的 Kubernetes 集群支持发现 Prometheus 指标。收集了 Prometheus 计数器、计量表和汇总指标类型。对直方图指标的支持计划在未来的版本中推出。

对于 Amazon ECS 和 Amazon EKS 集群，EC2 和 Fargate 启动类型均受支持。Container Insights 会自动从多个工作负载收集指标，您可以将其配置为从任何工作负载收集指标。

您可以采用 Prometheus 作为开源和开放标准方法，在 CloudWatch 中摄取自定义指标。具有 Prometheus 支持的 CloudWatch 代理可以发现并收集 Prometheus 指标，以便监控应用程序性能下降和故障，并更快地进行故障排除和发出告警。这也减少了改进可观测性所需的监控工具数量。

Container Insights Prometheus 支持涉及按用量付费的指标和日志，包括收集、存储和分析。有关更多信息，请参阅 Amazon CloudWatch 定价。

某些工作负载的预构建控制面板

Container Insights Prometheus 解决方案包括针对本节中列出的常见工作负载的预构建控制面板。有关这些工作负载的示例配置，请参阅 （可选）为 Prometheus 指标测试设置示例容器化 Amazon ECS 工作负载 和 （可选）为 Prometheus 指标测试设置示例容器化 Amazon EKS 工作负载。

您还可以配置 Container Insights 以通过编辑代理配置文件从其他容器化服务和应用程序收集 Prometheus 指标。

带有适用于在 Amazon EC2 实例上运行的 Amazon EKS 集群和 Kubernetes 集群的预构建控制面板的工作负载：

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

带有适用于 Amazon ECS 集群的预构建控制面板的工作负载：

- AWS App Mesh
- Java/JMX
- NGINX
- NGINX Plus

在亚马逊云服务器集群上设置和配置 Prometheus 指标收集 在 Amazon ECS 集群上设置和配置 Prometheus 指标收集

要从 Amazon ECS 集群收集 Prometheus 指标，您可以使用 CloudWatch 代理作为收集器或使用 AWS Distro for OpenTelemetry 收集器。有关使用 AWS Distro for OpenTelemetry 收集器的信息，请参阅 https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus。

下面几节介绍如何使用 CloudWatch 代理作为收集器来检索 Prometheus 指标。您可以在运行 Amazon ECS 的集群上安装带有 Prometheus 监控功能的 CloudWatch 代理，还可以选择配置该代理以抓取其他目标。这些部分还提供用于设置示例工作负载的可选教程，以使用 Prometheus 监控进行测试。

Amazon ECS 上的 Container Insights 支持 Prometheus 指标的以下启动类型和网络模式组合：

Amazon ECS 启动类型	支持的网络模式
EC2 (Linux)	网桥、主机和 awsvpc
Fargate	awsvpc

VPC 安全组要求

Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。

CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

主题

- 在 Amazon ECS 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理
- 抓取其他 Prometheus 源并导入这些指标
- （可选）为 Prometheus 指标测试设置示例容器化 Amazon ECS 工作负载

在 Amazon ECS 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理

本节介绍如何在运行 Amazon ECS 的集群中设置带有 Prometheus 监控功能的 CloudWatch 代理。执行此操作后，代理会自动抓取并导入该集群中运行的以下工作负载的指标。

- AWS App Mesh

- Java/JMX

您还可以将代理配置为从其他 Prometheus 工作负载和源抓取及导入指标。

设置 IAM 角色

您需要两个 IAM 角色来执行 CloudWatch 代理任务定义。如果您指定 AWS CloudFormation 堆栈中的 **`CreateIAMRoles=True`** 让 Container Insights 为您创建这些角色，则将使用正确的权限创建这些角色。如果要自己创建或使用现有角色，则需要以下角色和权限。

- CloudWatch 代理 ECS 任务角色 – CloudWatch 代理容器使用此角色。它必须包括 CloudWatchAgentServerPolicy 策略以及包含以下只读权限的客户管理策略：
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:ListServices`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeServices`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`
- CloudWatch 代理 ECS 任务执行角色 – 这是 Amazon ECS 启动和执行容器所需的角色。确保您的任务执行角色附加了 AmazonSSMReadOnlyAccess、AmazonECSTaskExecutionRolePolicy 和 CloudWatchAgentServerPolicy 策略。如果您需要存储更敏感的数据以供 Amazon ECS 使用，请参阅指定敏感数据以了解更多信息。

使用 AWS CloudFormation 安装带有 Prometheus 监控功能的 CloudWatch 代理

您使用 AWS CloudFormation 为 Amazon ECS 集群安装带有 Prometheus 监控功能的 CloudWatch 代理。以下列表显示了您将在 AWS CloudFormation 模板中使用的参数。

- ECSClusterName – 指定目标 Amazon ECS 集群。
- CreateIAMRoles – 指定 **`True`** 为 Amazon ECS 任务角色和 Amazon ECS 任务执行角色创建新角色。指定 **`False`** 重新使用现有角色。
- TaskRoleName – 如果您为 CreateIAMRoles 指定 **`True`**，这会指定用于新 Amazon ECS 任务角色的名称。如果您为 CreateIAMRoles 指定 **`False`**，这会指定要用作 Amazon ECS 任务角色的现有角色。

- ExecutionRoleName – 如果您为 CreateIAMRoles 指定 **True**，这会指定用于新 Amazon ECS 任务执行角色的名称。如果您为 CreateIAMRoles 指定 **False**，这会指定要用作 Amazon ECS 任务执行角色的现有角色。
- ECSNetworkMode – 如果您使用 EC2 启动类型，请在此处指定网络模式。必须是 **bridge** 或 **host**。
- ECSLaunchType – 指定 **fargate** 或 **EC2**。
- SecurityGroupID – 如果 ECSNetworkMode 是 **awsvpc**，请在此处指定安全组 ID。
- SubnetID – 如果 ECSNetworkMode 是 **awsvpc**，请在此处指定子网 ID。

命令示例

本节包含在各种情况下安装带有 Prometheus 监控功能的 Container Insights 的 AWS CloudFormation 示例命令。

在桥式网络模式下为 Amazon ECS 集群创建 AWS CloudFormation 堆栈

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

在主机网络模式下为 Amazon ECS 集群创建 AWS CloudFormation 堆栈

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name


curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

在 awsvpc 网络模式下为 Amazon ECS 集群创建 AWS CloudFormation 堆栈

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

在 awsvpc 网络模式下为 Fargate 集群创建 AWS CloudFormation 堆栈

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                 ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                 ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
```

```
                ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

AWS CloudFormation 堆栈创建的 AWS 资源

下表列出了 AWS 资源，该资源在 Amazon ECS 集群上使用 AWS CloudFormation 设置 带有 Prometheus 监控功能的 Container Insights 时创建。

资源类型	资源名称	注释
AWS::SSM::Parameter	AmazonCloudWatch-CWAgentConfig-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	这是具有默认 App Mesh 和 Java/JMX 嵌入式指标格式定义的 CloudWatch 代理。
AWS::SSM::Parameter	AmazonCloudWatch-Prometheus ConfigName-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	这是 Prometheus 抓取配置。
AWS::IAM::Role	$ECS_TASK_ROLE_NAME。	Amazon ECS 任务角色。这仅在您为 `CREATE_IAM_ROLES` 指定 **`True`** 时创建。
AWS::IAM::Role	${ECS_EXECUTION_ROLE_NAME}	Amazon ECS 任务执行角色。这仅在您为 `CREATE_IAM_ROLES` 指定 **`True`** 时创建。
AWS::ECS::TaskDefinition	cwagent-prometheus-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	
AWS::ECS::Service	cwagent-prometheus-replica-service-$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	

使用 Prometheus 监控功能删除 CloudWatch 代理的 AWS CloudFormation 堆栈

要从 Amazon ECS 集群中删除 CloudWatch 代理，请输入以下命令。

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name

aws cloudformation delete-stack \
--stack-name ${CLOUDFORMATION_STACK_NAME} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

抓取其他 Prometheus 源并导入这些指标

具有 Prometheus 监控功能的 CloudWatch 代理需要两种配置来抓取 Prometheus 指标。一个用于 Prometheus 文档中 <scrape_config> 记录的标准 Prometheus 配置。另一种配置是 CloudWatch 代理配置文件。

对于 Amazon ECS 集群，配置通过 Amazon ECS 任务定义中的密钥与 AWS Systems Manager 的 Parameter Store 集成：

- 密钥 PROMETHEUS_CONFIG_CONTENT 用于 Prometheus 抓取配置。
- 密钥 CW_CONFIG_CONTENT 用于 CloudWatch 代理配置。

要抓取其他 Prometheus 指标源并将这些指标导入 CloudWatch，您需要修改 Prometheus 抓取配置和 CloudWatch 代理配置，然后使用更新的配置重新部署代理。

VPC 安全组要求

Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。

CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

Prometheus 抓取配置

CloudWatch 代理支持标准的 Prometheus 抓取配置，如 Prometheus 文档中的 <scrape_config> 所述。您可以编辑此部分以更新此文件中已有的配置，并添加其他 Prometheus 抓取目标。默认情况下，示例配置文件包含以下全局配置行：

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- scrape_interval – 定义抓取目标的频率。
- scrape_timeout – 定义抓取请求超时之前的等待时间。

您还可以在作业级别为这些设置定义不同的值，以覆盖全局配置。

Prometheus 抓取任务

CloudWatch 代理 YAML 文件已配置了一些默认的抓取任务。例如，在 Amazon ECS 的 YAML 文件（例如 `cwagent-ecs-prometheus-metric-for-bridge-host.yaml`）中，默认抓取任务在 `ecs_service_discovery` 部分中配置。

```
"ecs_service_discovery": {
                  "sd_frequency": "1m",
                  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
                  "docker_label": {
                  },
                  "task_definition_list": [
                    {
                      "sd_job_name": "ecs-appmesh-colors",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition\/.*-
ColorTeller-(white):[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    },
                    {
                      "sd_job_name": "ecs-appmesh-gateway",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition/.*-
ColorGateway:[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    }
                  ]
                }
```

这些默认目标中的每个目标都会被抓取，并使用嵌入式指标格式在日志事件中将指标发送到 CloudWatch。有关更多信息，请参阅 在日志中嵌入指标。

来自 Amazon ECS 集群的日志事件存储在 /aws/ecs/containerinsights/***cluster_name***/prometheus 日志组中。

每个抓取作业都包含在此日志组中的不同日志流中。

要添加新的抓取目标，请在 YAML 文件 `ecs_service_discovery` 部分下的 `task_definition_list` 部分中添加一个新条目，然后重新启动代理。有关此过程的示例，请参阅添加新 Prometheus 抓取目标的教程：Prometheus API 服务器指标。

Prometheus 的 CloudWatch 代理配置

CloudWatch 代理配置文件在 `metrics_collected` 下面有一个 `prometheus` 部分用于 Prometheus 抓取配置。它包含以下配置选项：

- cluster_name – 指定要在日志事件中添加为标签的集群名称。该字段是可选的。如果省略此项，代理可以检测 Amazon ECS 集群名称。
- log_group_name – 指定已抓取 Prometheus 指标的日志组名称。该字段是可选的。如果省略此项，CloudWatch 会将 /aws/ecs/containerinsights/***cluster_name***/prometheus 用于来自 Amazon ECS 集群的日志。
- prometheus_config_path – 指定 Prometheus 抓取配置文件路径。如果此字段的值以 `env:` 开头，Prometheus 抓取配置文件内容将从容器的环境变量中检索。请勿更改此字段。
- ecs_service_discovery – 是指定 Amazon ECS Prometheus 目标自动发现功能配置的部分。支持两种模式来发现 Prometheus 目标：根据容器的 docker 标签的发现或根据 Amazon ECS 任务定义 ARN 正则表达式的发现。您可以同时使用这两种模式，CloudWatch 代理将根据 {private_ip}:{port}/{metrics_path} 对发现的目标进行重复去除。

  `ecs_service_discovery` 部分包含以下字段：

  - `sd_frequency` 是发现 Prometheus 导出器的频率。指定数字和单位后缀。例如，`1m` 表示每分钟一次或 `30s` 表示每 30 秒一次。有效的单位后缀为 `ns`、`us`、`ms`、`s`、`m` 和 `h`。

    该字段是可选的。默认值为 60 秒（1 分钟）。
  - `sd_target_cluster` 是用于自动发现的目标 Amazon ECS 集群名称。该字段是可选的。默认名称为安装 CloudWatch 代理的 Amazon ECS 集群的名称。
  - `sd_cluster_region` 是目标 Amazon ECS 集群的区域。该字段是可选的。默认区域为安装 CloudWatch 代理的 Amazon ECS 集群的区域。
  - `sd_result_file` 是 Prometheus 目标结果的 YAML 文件的路径。Prometheus 抓取配置将引用此文件。

- docker_label 是可选部分，您可以使用它来指定基于 docker 标签的服务发现的配置。如果省略此部分，则不会使用基于 docker 标签的发现。此部分包含以下字段：
  - sd_port_label 是容器的 docker 标签名称，用于指定 Prometheus 指标的容器端口。默认值为 ECS_PROMETHEUS_EXPORTER_PORT。如果容器没有此 docker 标签，CloudWatch 代理将跳过它。
  - sd_metrics_path_label 是容器的 docker 标签名称，用于指定 Prometheus 指标路径。默认值为 ECS_PROMETHEUS_METRICS_PATH。如果容器没有此 docker 标签，则代理会采用默认路径 /metrics。
  - sd_job_name_label 是容器的 docker 标签名称，用于指定 Prometheus 抓取任务名称。默认值为 job。如果容器没有此 docker 标签，CloudWatch 代理会使用 Prometheus 抓取配置中的任务名称。
- task_definition_list 是可选部分，可用于指定基于任务定义的服务发现的配置。如果省略此部分，则不会使用基于任务定义的发现。此部分包含以下字段：
  - sd_task_definition_arn_pattern 是用于指定要发现的 Amazon ECS 任务定义的模式。这是正则表达式。
  - sd_metrics_ports 列出 Prometheus 指标的 containerPort。用分号分隔 containerPorts。
  - sd_container_name_pattern 指定 Amazon ECS 任务容器名称。这是正则表达式。
  - sd_metrics_path 指定 Prometheus 指标路径。如果省略此项，代理会采用默认路径 /metrics
  - sd_job_name 指定 Prometheus 抓取任务名称。如果省略此字段，CloudWatch 代理会使用 Prometheus 抓取配置中的任务名称。
- service_name_list_for_tasks 是可选部分，可用于指定基于服务名称的发现的配置。如果省略此部分，则不会使用基于服务名称的发现。此部分包含以下字段：
  - sd_service_name_pattern 是用于指定要在其中发现任务的 Amazon ECS 服务的模式。这是正则表达式。
  - sd_metrics_ports 列出 Prometheus 指标的 containerPort。用分号分隔多个 containerPorts。
  - sd_container_name_pattern 指定 Amazon ECS 任务容器名称。这是正则表达式。
  - sd_metrics_path 指定 Prometheus 指标路径。如果省略此项，代理会采用默认路径 /metrics。
  - sd_job_name 指定 Prometheus 抓取任务名称。如果省略此字段，CloudWatch 代理会使用 Prometheus 抓取配置中的任务名称。

- metric_declaration – 是指定要生成的采用嵌入式指标格式的日志数组的部分。默认情况下，CloudWatch 代理从中进行导入的每个 Prometheus 源都有 metric_declaration 部分。这些部分各包括以下字段：
  - label_matcher 是一个正则表达式，用于检查 source_labels 中列出的标签的值。匹配的指标将启用，以包含在发送到 CloudWatch 的嵌入式指标格式中。

    如果您在 source_labels 中指定了多个标签，我们建议您不要在 label_matcher 的正则表达式中使用 ^ 或 $ 字符。
  - source_labels 指定由 label_matcher 行检查的标签的值。
  - label_separator 指定要在 label_matcher 行中使用的分隔符（如果指定了多个 source_labels）。默认为 ;。您可以在以下示例中看到 label_matcher 行中使用的此默认值。
  - metric_selectors 是一个正则表达式，用于指定要收集并发送到 CloudWatch 的指标。
  - dimensions 是要用作每个选定指标的 CloudWatch 维度的标签列表。

请参阅以下 metric_declaration 示例。

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system$",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

此示例配置嵌入式指标格式部分，以便在满足以下条件时作为日志事件发送：

- Service 的值包含 node-exporter 或 kube-dns。
- Namespace 的值为 kube-system。
- Prometheus 指标 coredns_dns_request_type_count_total 同时包含 Service 和 Namespace 标签。

发送的日志事件包括以下突出显示的部分：

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
   "Service":"kube-dns",
   "coredns_dns_request_type_count_total":2562,
   "eks_amazonaws_com_component":"kube-dns",
   "instance":"192.168.61.254:9153",
   "job":"kubernetes-service-endpoints",
   ...
}
```

Amazon ECS 集群上自动发现的详细指南

Prometheus 提供了数十种动态服务发现机制，如 <scrape_config> 中所述。但是，Amazon ECS 没有内置服务发现。CloudWatch 代理添加了此机制。

启用 Amazon ECS Prometheus 服务发现后，CloudWatch 代理会定期对 Amazon ECS 和 Amazon EC2 前端进行以下 API 调用，以检索目标 ECS 集群中正在运行的 ECS 任务的元数据。

```
EC2:DescribeInstances
ECS:ListTasks
ECS:ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
```

```
ECS:DescribeTaskDefinition
```

CloudWatch 代理使用元数据来扫描 ECS 集群内的 Prometheus 目标。CloudWatch 代理支持三种服务发现模式：

- 基于容器 docker 标签的服务发现
- 基于 ECS 任务定义 ARN 正则表达式的服务发现
- 基于 ECS 服务名称正则表达式的服务发现

所有模式均可同时使用。CloudWatch 代理会根据 `{private_ip}:{port}/{metrics_path}` 对发现的目标进行重复去除。

所有发现的目标均写入由 CloudWatch 代理容器内的 `sd_result_file` 配置字段指定的结果文件。以下为示例结果文件：

```
- targets:
  - 10.6.1.95:32785
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT: "9406"
    ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
    TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
    TaskRevision: "7"
    VpcId: vpc-01234567890
    container_name: demo-jar-ec2-bridge-dynamic-port
    job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
    ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
```

```
    TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
    TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
    TaskRevision: "12"
    VpcId: vpc-01234567890
    container_name: demo-tomcat-ec2-bridge-mapped-port
    job: demo-tomcat-ec2-bridge-mapped-port
```

您可以直接将此结果文件与基于 Prometheus 文件的服务发现集成。有关基于 Prometheus 文件的服务发现的更多信息，请参阅 <file_sd_config>。

假设将结果文件写入 `/tmp/cwagent_ecs_auto_sd.yaml`。以下 Prometheus 抓取配置将使用它。

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

CloudWatch 代理还会为发现的目标添加以下附加标签。

- `container_name`
- `TaskDefinitionFamily`
- `TaskRevision`
- `TaskGroup`
- `StartedBy`
- `LaunchType`
- `job`
- `__metrics_path__`
- Docker 标签

当集群具有 EC2 启动类型时，会添加以下三个标签。

- `InstanceType`
- `VpcId`
- `SubnetId`

> **Note**
>
> 与正则表达式 `[a-zA-Z_][a-zA-Z0-9_]*` 不匹配的 Docker 标签将被筛选掉。这与 Prometheus 文档中配置文件的 `label_name` 中列出的 Prometheus 惯例相匹配。

ECS 服务发现配置示例

本节包括演示 ECS 服务发现的示例。

示例 1

```
"ecs_service_discovery": {
  "sd_frequency": "1m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
  }
}
```

此示例启用基于 docker 标签的服务发现。CloudWatch 代理将每分钟查询一次 ECS 任务的元数据，并将发现的目标写入 CloudWatch 代理容器内的 `/tmp/cwagent_ecs_auto_sd.yaml` 文件中。

`docker_label` 部分中 `sd_port_label` 的默认值为 `ECS_PROMETHEUS_EXPORTER_PORT`。如果 ECS 任务中任何正在运行的容器具有 `ECS_PROMETHEUS_EXPORTER_PORT` docker 标签，CloudWatch 代理将其值作为 `container port` 来扫描容器的所有公开端口。如果匹配，则使用映射的主机端口和容器的私有 IP，以 `private_ip:host_port` 格式来构建 Prometheus 导出器目标。

`docker_label` 部分中 `sd_metrics_path_label` 的默认值为 `ECS_PROMETHEUS_METRICS_PATH`。如果容器具有此 docker 标签，则其值将用作 `__metrics_path__`。如果容器没有此标签，则使用默认值 `/metrics`。

`docker_label` 部分中 `sd_job_name_label` 的默认值为 `job`。如果容器具有此 docker 标签，则其值将作为一个目标标签附加，以替换 Prometheus 配置中指定的默认任务名称。此 docker 标签的值用作 CloudWatch Logs 日志组中的日志流名称。

示例 2

```
"ecs_service_discovery": {
  "sd_frequency": "15s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
```

```
  "docker_label": {
    "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
    "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
  }
}
```

此示例启用基于 docker 标签的服务发现。CloudWatch 代理将每 15 秒查询一次 ECS 任务的元数据，并将发现的目标写入 CloudWatch 代理容器内的 /tmp/cwagent_ecs_auto_sd.yaml 文件中。带有 ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A 的 docker 标签的容器将被扫描。docker 标签 ECS_PROMETHEUS_JOB_NAME 的值用作任务名称。

示例 3

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "task_definition_list": [
    {
      "sd_job_name": "java-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9404; 9406",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
    },
    {
      "sd_job_name": "envoy-prometheus",
      "sd_metrics_path": "/stats/prometheus",
      "sd_container_name_pattern": "^envoy$",
      "sd_metrics_ports": "9901",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
  ]
}
```

此示例启用基于 ECS 任务定义 ARN 正则表达式的服务发现。CloudWatch 代理将每五分钟查询一次 ECS 任务的元数据，并将发现的目标写入 CloudWatch 代理容器内的 /tmp/cwagent_ecs_auto_sd.yaml 文件中。

定义了两个任务定义 ARN 正则表达式部分：

- 对于第一部分，过滤 ECS 任务定义 ARN 中的 ECS 任务以及 javajmx，以进行容器端口扫描。如果这些 ECS 任务中的容器在 9404 或 9406 上公开容器端口，则映射的主机端口和容器的私有 IP 会用于创建 Prometheus 导出器目标。sd_metrics_path 的值将 __metrics_path__

设置为 /metrics。因此 CloudWatch 代理将从 private_ip:host_port/metrics 中抓取 Prometheus 指标，抓取的指标发送到日志组 /aws/ecs/containerinsights/cluster_name/prometheus 中 CloudWatch Logs 中的 java-prometheus 日志流。

- 对于第二部分，过滤 ECS 任务定义 ARN 中带有 appmesh 的 ECS 任务以及 :23 的 version，以进行容器端口扫描。对于在 9901 上公开容器端口的、名称为 envoy 的容器，映射的主机端口以及容器的私有 IP 用于创建 Prometheus 导出器目标。这些 ECS 任务中的值会公开 9404 或 9406 上的容器端口，映射的主机端口以及容器的私有 IP 用于创建 Prometheus 导出器目标。sd_metrics_path 的值将 __metrics_path__ 设置为 /stats/prometheus。因此，CloudWatch 代理将从 private_ip:host_port/stats/prometheus 中抓取 Prometheus 指标，并将抓取的指标发送到日志组 /aws/ecs/containerinsights/cluster_name/prometheus 中的 CloudWatch Logs 中的 envoy-prometheus 日志流。

示例 4。

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "service_name_list_for_tasks": [
    {
      "sd_job_name": "nginx-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9113",
      "sd_service_name_pattern": "^nginx-.*"
    },
    {
      "sd_job_name": "haproxy-prometheus",
      "sd_metrics_path": "/stats/metrics",
      "sd_container_name_pattern": "^haproxy$",
      "sd_metrics_ports": "8404",
      "sd_service_name_pattern": ".*haproxy-service.*"
    }
  ]
}
```

本示例启用基于 ECS 服务名称正则表达式的服务发现。CloudWatch 代理将每五分钟查询一次 ECS 服务的元数据，并将发现的目标写入 CloudWatch 代理容器内的 /tmp/cwagent_ecs_auto_sd.yaml 文件中。

定义了两个服务名称正则表达式部分：

- 对于第一部分，过滤与 ECS 服务关联的 ECS 任务（名称与正则表达式 ^nginx-.* 相匹配），以进行容器端口扫描。如果这些 ECS 任务中的容器在 9113 上公开容器端口，则映射的主机端口和容器的私有 IP 将用于创建 Prometheus 导出器目标。sd_metrics_path 的值将 __metrics_path__ 设置为 /metrics。因此，CloudWatch 代理将从 private_ip:host_port/metrics 中抓取 Prometheus 指标，并将抓取的指标发送到日志组 /aws/ecs/containerinsights/cluster_name/prometheus 中 CloudWatch Logs 中的 nginx-prometheus 日志流。
- 或第二部分，过滤与 ECS 服务关联的 ECS 任务（名称与正则表达式 .*haproxy-service.* 匹配），以进行容器端口扫描。对于在 8404 上公开容器端口、名称为 haproxy 的容器，映射的主机端口以及容器的私有 IP 用于创建 Prometheus 导出器目标。sd_metrics_path 的值将 __metrics_path__ 设置为 /stats/metrics。因此，CloudWatch 代理将从 private_ip:host_port/stats/metrics 中抓取 Prometheus 指标，并将抓取的指标发送到日志组 /aws/ecs/containerinsights/cluster_name/prometheus 中 CloudWatch Logs 中的 haproxy-prometheus 日志流。

示例 5

```
"ecs_service_discovery": {
  "sd_frequency": "1m30s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
    "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
    "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
  }
  "task_definition_list": [
    {
      "sd_metrics_ports": "9150",
      "sd_task_definition_arn_pattern": "*memcached.*"
    }
  ]
}
```

本示例启用了两种 ECS 服务发现模式。CloudWatch 代理将每 90 秒查询一次 ECS 任务的元数据，并将发现的目标写入 CloudWatch 代理容器内的 /tmp/cwagent_ecs_auto_sd.yaml 文件中。

对于基于 docker 的服务发现配置：

- 过滤带有 docker 标签 `MY_PROMETHEUS_EXPORTER_PORT_LABEL` 的 ECS 任务，以进行 Prometheus 端口扫描。目标 Prometheus 容器端口由 label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` 的值指定。
- docker 标签 `MY_PROMETHEUS_EXPORTER_PORT_LABEL` 的值用于 `__metrics_path__`。如果容器没有此 docker 标签，则使用默认值 `/metrics`。
- docker 标签 `MY_PROMETHEUS_EXPORTER_PORT_LABEL` 的值用作任务标签。如果容器没有此 docker 标签，则使用 Prometheus 配置中定义的任务名称。

对于基于 ECS 任务定义 ARN 正则表达式的服务发现配置：

- 过滤 ECS 任务定义 ARN 中带 `memcached` 的 ECS 任务，以进行容器端口扫描。根据 `sd_metrics_ports` 定义，目标 Prometheus 容器端口是 9150。使用默认指标路径 `/metrics`。使用 Prometheus 配置中定义的任务名称。

（可选）为 Prometheus 指标测试设置示例容器化 Amazon ECS 工作负载

要测试 CloudWatch Container Insights 中的 Prometheus 指标支持，您可设置以下一个或多个容器化工作负载。具有 Prometheus 支持的 CloudWatch 代理会自动从这些工作负载中的每一个收集指标。要查看默认情况下收集的指标，请参阅 CloudWatch 代理收集的 Prometheus 指标。

主题

- Amazon ECS 集群的示例 App Mesh 工作负载
- Amazon ECS 集群的示例 Java/JMX 工作负载
- Amazon ECS 集群的示例 NGINX 工作负载
- Amazon ECS 集群的示例 NGINX Plus 工作负载
- 添加新 Prometheus 抓取目标的教程：Amazon ECS 上的 Memcached
- 在 Amazon ECS Fargate 上抓取 Redis OSS Prometheus 指标的教程

Amazon ECS 集群的示例 App Mesh 工作负载

要从 Amazon ECS 的示例 Prometheus 工作负载收集指标，您必须在集群中运行 Container Insights。有关安装 Container Insights 的信息，请参阅 在 Amazon ECS 上设置 Container Insights。

首先，按照本演练在 Amazon ECS 集群上部署示例颜色应用程序。完成后，您将在端口 9901 上公开 App Mesh Prometheus 指标。

接下来，按照以下步骤在安装颜色应用程序的同一 Amazon ECS 集群上，安装带有 Prometheus 监控功能的 CloudWatch 代理。本节中的步骤以桥式网络模式安装 CloudWatch 代理。

您在演练中设置的环境变量 ENVIRONMENT_NAME、AWS_PROFILE 和 AWS_DEFAULT_REGION也将在以下步骤中使用。

安装带有 Prometheus 监控功能的 CloudWatch 代理以进行测试

1. 通过输入以下命令，下载 AWS CloudFormation 模板。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. 通过输入以下命令，设置网络模式。

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. 输入以下命令以创建 AWS CloudFormation 堆栈。

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-TaskRole-${ECS_CLUSTER_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-ExecutionRole-${ECS_CLUSTER_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

4. （可选）创建 AWS CloudFormation 堆栈后，您会看到 CREATE_COMPLETE 消息。如果要在看到该消息之前检查状态，请输入以下命令。

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
```

```
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

故障排除

演练中的步骤使用 jq 来分析 AWS CLI 的输出结果。有关安装 jq 代理的更多信息，请参阅 jq。使用以下命令将 AWS CLI 的默认输出格式设置为 JSON，以便 jq 可以对其进行正确分析。

```
$ aws configure
```

当响应变为 `Default output format` 时，输入 **`json`**。

使用 Prometheus 监控功能卸载 CloudWatch 代理

完成测试后，输入以下命令以通过 AWS CloudFormation 删除堆栈来卸载 CloudWatch 代理。

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Amazon ECS 集群的示例 Java/JMX 工作负载

JMX Exporter 是 Prometheus 的官方导出程序，可以将 JMX MBeans 作为 Prometheus 指标进行抓取和公开。有关详细信息，请参阅 prometheus/jmx_exporter。

支持 Prometheus 的 CloudWatch 代理根据 Amazon ECS 集群中的服务发现配置抓取 Java/JMX Prometheus 指标。您可以将 JMX Exporter 配置为在不同端口或 metrics_path 上公开指标。如果您更改了端口或路径，请更新 CloudWatch 代理配置中的默认 `ecs_service_discovery` 部分。

要从 Amazon ECS 的示例 Prometheus 工作负载收集指标，您必须在集群中运行 Container Insights。有关安装 Container Insights 的信息，请参阅 在 Amazon ECS 上设置 Container Insights。

为 Amazon ECS 集群安装 Java/JMX 示例工作负载

1. 按照这些部分中的步骤创建 Docker 镜像。

   - 示例：具有 Prometheus 指标的 Java Jar 应用程序 Docker 镜像

- 示例：具有 Prometheus 指标的 Apache Tomcat Docker 镜像

2. 在 Amazon ECS 任务定义文件中指定以下两个 docker 标签。然后，您可以将任务定义作为集群中的 Amazon ECS 服务或 Amazon ECS 任务运行。

   - 将 ECS_PROMETHEUS_EXPORTER_PORT 设置为指向公开 Prometheus 指标的 containerPort。
   - 将 Java_EMF_Metrics 设置为 true。CloudWatch 代理使用此标志在日志事件中生成嵌入的指标格式。

   以下是 示例：

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
        }
      ],
      "dockerLabels": {
        "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
        "Java_EMF_Metrics": "true"
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"  ],
  "cpu": "256",
  "memory": "512"
  }
```

AWS CloudFormation 模板中 CloudWatch 代理的默认设置同时启用基于 docker 标签的服务发现和基于任务定义 ARN 的服务发现。要查看这些默认设置，请参阅 CloudWatch 代理 YAML 配置文件的第

65 行。带有 ECS_PROMETHEUS_EXPORTER_PORT 标签的容器将根据 Prometheus 抓取的指定容器端口自动发现。

CloudWatch 代理的默认设置在同一文件的第 112 行也有 Java/JMX metric_declaration 设置。目标容器的所有 docker 标签将作为附加标签添加到 Prometheus 指标中并发送到 CloudWatch Logs。对于带有 docker 标签 Java_EMF_Metrics="true" 的 Java/JMX 容器，将生成嵌入式指标格式。

Amazon ECS 集群的示例 NGINX 工作负载

NGINX Prometheus 导出器可以抓取和公开 NGINX 数据作为 Prometheus 指标。此示例将导出器与适用于 Amazon ECS 的 NGINX 反向代理服务结合使用。

有关 NGINX Prometheus 导出器的更多信息，请参阅 Github 上的 nginx-prometheus-exporter。有关 NGINX 反向代理的更多信息，请参阅 Github 上的 ecs-nginx-reverse-proxy。

支持 Prometheus 的 CloudWatch 代理根据 Amazon ECS 集群中的服务发现配置抓取 NGINX Prometheus 指标。您可以将 NGINX Prometheus Exporter 配置为在不同端口或路径上公开指标。如果您更改端口或路径，请更新 CloudWatch 代理配置文件中的 ecs_service_discovery 部分。

为 Amazon ECS 集群安装 NGINX 反向代理示例工作负载

按照以下步骤安装 NGINX 反向代理示例工作负载。

创建 Docker 镜像

为 NGINX 反向代理示例工作负载创建 Docker 镜像

1. 从 NGINX 反向代理存储库下载以下文件夹：https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/。
2. 查找 app 目录并从该目录构建镜像：

```
docker build -t web-server-app ./path-to-app-directory
```

3. 为 NGINX 构建自定义镜像。首先，创建一个包含以下两个文件的目录：
   - 示例 Dockerfile：

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

   - 一个 nginx.conf 文件，修改自 https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/：

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  server{
    listen 8080;
    location /stub_status {
        stub_status   on;
    }
  }

  server {
    listen 80;

    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://app:3000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_cache_bypass $http_upgrade;
    }
  }
}
```

> **Note**
>
> `stub_status` 必须在将 `nginx-prometheus-exporter` 配置为从中抓取指标的同一端口上启用。在我们的示例任务定义中，将 `nginx-prometheus-exporter` 配置为从端口 8080 抓取指标。

4. 从新目录中的文件构建镜像：

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. 将新镜像上传到镜像存储库以供日后使用。

创建任务定义以在 Amazon ECS 中运行 NGINX 和 web 服务器应用程序

接下来，设置任务定义。

此任务定义启用 NGINX Prometheus 指标的收集和导出。NGINX 容器跟踪来自应用程序的输入，并将该数据公开到端口 8080，如 `nginx.conf` 中所设置。NGINX prometheus 导出器容器抓取这些指标，并将其发布到端口 9113，以在 CloudWatch 中使用。

为 NGINX 示例 Amazon ECS 工作负载设置任务定义

1. 使用以下内容创建任务定义 JSON 文件。将 *your-customized-nginx-iamge* 替换为您自定义的 NGINX 镜像的镜像 URI，并将 *your-web-server-app-image* 替换为您 web 服务器应用程序镜像的镜像 URI。

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
```

```
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.scrape-uri",
        "http://nginx:8080/stub_status"
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-sample-stack"
}
```

2. 使用以下命令注册任务定义。

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. 通过输入以下命令创建服务以运行任务：

确保不要更改服务名称。我们将使用配置来运行 CloudWatch 代理服务，该配置使用启动它们的服务的名称模式来搜索任务。例如，要让 CloudWatch 代理查找此命令启动的任务，您可以将 `sd_service_name_pattern` 的值指定为 `^nginx-service$`。下一部分将提供更多详细信息。

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-service \
 --task-definition nginx-sample-stack:1 \
 --desired-count 1
```

配置 CloudWatch 代理以抓取 NGINX Prometheus 指标

最后一步是配置 CloudWatch 代理以抓取 NGINX 指标。在此示例中，CloudWatch 代理通过服务名称模式和端口 9113 发现任务，导出器在该端口公开 NGINX 的 prometheus 指标。发现任务且指标可用后，CloudWatch 代理开始将收集的指标发布到日志流 nginx-prometheus-exporter。

配置 CloudWatch 代理以抓取 NGINX 指标

1. 通过输入以下命令，下载必要 YAML 文件的最新版本。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. 使用文本编辑器打开文件，然后在 `resource:CWAgentConfigSSMParameter` 部分的 `value` 密钥中查找完整的 CloudWatch 代理配置。然后，在 `ecs_service_discovery` 部分中，添加以下 `service_name_list_for_tasks` 部分。

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-service$"
   }
],
```

3. 在同一文件中，在 metric_declaration 部分中添加以下部分以允许 NGINX 指标。请务必遵循现有的缩进模式。

```
{
  "source_labels": ["job"],
  "label_matcher": ".*nginx.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginx_.*$"
  ]
},
```

4. 如果您尚未在此集群中部署 CloudWatch 代理，请跳至步骤 8。

   如果您已经使用 AWS CloudFormation 将 CloudWatch 代理部署在 Amazon ECS 集群中，您可以通过输入以下命令来创建更改集：

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-scraping-support
```

5. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
6. 查看新创建的变更集 nginx-scraping-support。您会看到一项应用于 CWAgentConfigSSMParameter 资源的更改。通过输入以下命令，运行变更集并重新启动 CloudWatch 代理任务：

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. 等待大约 10 秒，然后输入以下命令。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. 如果您是首次在集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理，请输入以下命令。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

查看您的 NGINX 指标和日志

您现在可以查看正在收集的 NGINX 指标。

查看示例 NGINX 工作负载的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在运行集群的区域中，选择左侧导航窗格中的 Metrics（指标）。查找 ContainerInsights/Prometheus 命名空间以查看指标。
3. 要查看 CloudWatch Logs 事件，请在导航窗格中选择 Log groups（日志组）。这些事件位于日志流 *nginx-prometheus-exporter* 中的日志组 /aws/containerinsights/***your_cluster_name***/prometheus 中。

Amazon ECS 集群的示例 NGINX Plus 工作负载

NGINX Plus 是 NGINX 的商业版本。您必须拥有许可证方可使用它。有关更多信息，请参阅 NGINX Plus

NGINX Prometheus 导出器可以抓取和公开 NGINX 数据作为 Prometheus 指标。此示例将导出器与适用于 Amazon ECS 的 NGINX Plus 反向代理服务结合使用。

有关 NGINX Prometheus 导出器的更多信息，请参阅 Github 上的 nginx-prometheus-exporter。有关 NGINX 反向代理的更多信息，请参阅 Github 上的 ecs-nginx-reverse-proxy。

支持 Prometheus 的 CloudWatch 代理根据 Amazon ECS 集群中的服务发现配置抓取 NGINX Plus Prometheus 指标。您可以将 NGINX Prometheus Exporter 配置为在不同端口或路径上公开指标。如果您更改端口或路径，请更新 CloudWatch 代理配置文件中的 `ecs_service_discovery` 部分。

为 Amazon ECS 集群安装 NGINX Plus 反向代理示例工作负载

按照以下步骤安装 NGINX 反向代理示例工作负载。

创建 Docker 镜像

为 NGINX Plus 反向代理示例工作负载创建 Docker 镜像

1. 从 NGINX 反向代理存储库下载以下文件夹：https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/。
2. 查找 app 目录并从该目录构建镜像：

   ```
   docker build -t web-server-app ./path-to-app-directory
   ```

3. 为 NGINX Plus 构建自定义镜像。在为 NGINX Plus 构建镜像之前，您需要为您的 NGINX Plus 许可获取名为 `nginx-repo.key` 的密钥和 SSL 证书 `nginx-repo.crt`。创建目录并将您的 `nginx-repo.key` 和 `nginx-repo.crt` 文件存储在其中。

   在刚刚创建的目录中，创建以下两个文件：

- 使用以下内容创建示例 Dockerfile。此 docker 文件取自 https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image 上提供的示例文件。我们所做的重要更改是我们加载了一个名为 nginx.conf 的单独文件，该文件将在下一步中创建。

```
FROM debian:buster-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>“

# Define NGINX versions for NGINX Plus and NGINX Plus modules
# Uncomment this block and the versioned nginxPackages block in the main RUN
# instruction to install a specific release
# ENV NGINX_VERSION 21
# ENV NJS_VERSION 0.3.9
# ENV PKG_RELEASE 1~buster

# Download certificate and key from the customer portal (https://cs.nginx.com
 (https://cs.nginx.com/))
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# COPY nginx.conf /etc/ssl/nginx/nginx.conf

RUN set -x \
# Create nginx user/group first, to be consistent throughout Docker variants
&& addgroup --system --gid 101 nginx \
&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /
nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y ca-
certificates gnupg1 \
&& \
NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
hkp://keyserver.ubuntu.com:80 \
hkp://p80.pool.sks-keyservers.net:80 \
pgp.mit.edu (http://pgp.mit.edu/) \
; do \
echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
 "$NGINX_GPGKEY" && found=yes && break; \
```

```
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" &&
 exit 1; \
apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-
repo.crt\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-
repo.key\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/
apt/sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/
apt/sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log
```

```
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

- 一个修改自 https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx 的 nginx.conf 文件。

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  upstream backend {
    zone name 10m;
    server app:3000    weight=2;
    server app2:3000    weight=1;
  }

  server{
    listen 8080;
    location /api {
      api write=on;
    }
  }

  match server_ok {
    status 100-599;
  }

  server {
    listen 80;
    status_zone zone;
    # Nginx will reject anything not matching /api
```

```
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://backend;
      health_check uri=/lorem-ipsum match=server_ok;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_cache_bypass $http_upgrade;
    }
  }
}
```

4. 从新目录中的文件构建镜像：

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. 将新镜像上传到镜像存储库以供日后使用。

创建任务定义以在 Amazon ECS 中运行 NGINX Plus 和 web 服务器应用程序

接下来，设置任务定义。

此任务定义启用 NGINX Plus Prometheus 指标的收集和导出。NGINX 容器跟踪来自应用程序的输入，并将该数据公开到端口 8080，如 `nginx.conf` 中所设置。NGINX prometheus 导出器容器抓取这些指标，并将其发布到端口 9113，以在 CloudWatch 中使用。

为 NGINX 示例 Amazon ECS 工作负载设置任务定义

1. 使用以下内容创建任务定义 JSON 文件。将 *your-customized-nginx-plus-image* 替换为自定义 NGINX Plus 镜像的镜像 URI，并将 *your-web-server-app-image* 替换为 Web 服务器应用程序镜像的镜像 URI。

```
{
  "containerDefinitions": [
```

```
    {
      "name": "nginx",
      "image": "your-customized-nginx-plus-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app",
        "app2"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "app2",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.plus",
        "-nginx.scrape-uri",
         "http://nginx:8080/api"
    ],
    "links":[
      "nginx"
```

```
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-plus-sample-stack"
}
```

2. 注册任务定义：

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. 通过输入以下命令创建服务以运行任务：

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-plus-service \
 --task-definition nginx-plus-sample-stack:1 \
 --desired-count 1
```

确保不要更改服务名称。我们将使用配置来运行 CloudWatch 代理服务，该配置使用启动它们的服务的名称模式来搜索任务。例如，要让 CloudWatch 代理查找此命令启动的任务，您可以将 `sd_service_name_pattern` 的值指定为 `^nginx-plus-service$`。下一部分将提供更多详细信息。

配置 CloudWatch 代理以抓取 NGINX Plus Prometheus 指标

最后一步是配置 CloudWatch 代理以抓取 NGINX 指标。在此示例中，CloudWatch 代理通过服务名称模式和端口 9113 发现任务，导出器在该端口公开 NGINX 的 prometheus 指标。发现任务且指标可用后，CloudWatch 代理开始将收集的指标发布到日志流 nginx-prometheus-exporter。

配置 CloudWatch 代理以抓取 NGINX 指标

1. 通过输入以下命令，下载必要 YAML 文件的最新版本。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
bridge-host.yaml
```

2. 使用文本编辑器打开文件，然后在 `resource:CWAgentConfigSSMParameter` 部分的 `value` 密钥中查找完整的 CloudWatch 代理配置。然后，在 `ecs_service_discovery` 部分中，添加以下 `service_name_list_for_tasks` 部分。

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
   }
],
```

3. 在同一个文件中，在 `metric_declaration` 的部分中添加以下部分，以允许 NGINX Plus 指标。请务必遵循现有的缩进模式。

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginxplus_connections_accepted$",
    "^nginxplus_connections_active$",
    "^nginxplus_connections_dropped$",
    "^nginxplus_connections_idle$",
    "^nginxplus_http_requests_total$",
    "^nginxplus_ssl_handshakes$",
    "^nginxplus_ssl_handshakes_failed$",
    "^nginxplus_up$",
    "^nginxplus_upstream_server_health_checks_fails$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName",
 "upstream"]],
```

```
  "metric_selectors": [
    "^nginxplus_upstream_server_response_time$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_responses$",
    "^nginxplus_server_zone_responses$"
  ]
},
```

4. 如果您尚未在此集群中部署 CloudWatch 代理，请跳至步骤 8。

   如果您已经使用 AWS CloudFormation 将 CloudWatch 代理部署在 Amazon ECS 集群中，您可以通过输入以下命令来创建更改集：

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-plus-scraping-support
```

5. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
6. 查看新创建的变更集 nginx-plus-scraping-support。您会看到一项应用于 CWAgentConfigSSMParameter 资源的更改。通过输入以下命令，运行变更集并重新启动 CloudWatch 代理任务：

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. 等待大约 10 秒，然后输入以下命令。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. 如果您是首次在集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理，请输入以下命令。

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

查看您的 NGINX Plus 指标和日志

您现在可以查看正在收集的 NGINX Plus 指标。

查看示例 NGINX 工作负载的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在运行集群的区域中，选择左侧导航窗格中的 Metrics（指标）。查找 ContainerInsights/Prometheus 命名空间以查看指标。
3. 要查看 CloudWatch Logs 事件，请在导航窗格中选择 Log groups（日志组）。事件位于日志流 *nginx-plus-prometheus-exporter* 中的日志组 /aws/containerinsights/***your_cluster_name***/prometheus 中。

添加新 Prometheus 抓取目标的教程：Amazon ECS 上的 Memcached

本教程提供实践介绍，关于如何在 Amazon ECS 集群上使用 EC2 启动类型抓取示例 Memcached 应用程序的 Prometheus 指标。Memcached Prometheus 导出器目标将由 CloudWatch 代理通过基于 ECS 任务定义的服务发现自动发现。

Memcached 是一个通用型分布式内存缓存系统。它通常用于通过在 RAM 中缓存数据和对象来加速动态数据库驱动的网站，以减少必须读取的外部数据源（例如数据库或 API）的次数。有关更多信息，请参阅什么是 Memcached？

memchached_exporter（Apache 许可证 2.0）是 Prometheus 一款官方导出器。默认情况下，memcache_exporter 在 `/metrics.` 的端口 0.0.0.0:9150 上提供服务

本教程使用了以下两个 Docker Hub 存储库中的 Docker 镜像：

- Memcached
- prom/memcached-exporter

先决条件

要从 Amazon ECS 的示例 Prometheus 工作负载收集指标，您必须在集群中运行 Container Insights。有关安装 Container Insights 的信息，请参阅 在 Amazon ECS 上设置 Container Insights。

主题

- 设置 Amazon ECS EC2 集群环境变量
- 安装 Memcached 示例工作负载
- 配置 CloudWatch 代理以抓取 Memcached Prometheus 指标
- 查看您的 Memcached 指标

设置 Amazon ECS EC2 集群环境变量

设置 Amazon ECS EC2 集群环境变量

1. 请安装 Amazon ECS CLI（如果尚未安装）。有关更多信息，请参阅安装 Amazon ECS CLI。
2. 设置新的 Amazon ECS 集群名称和区域。例如：

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. （可选）如果您还没有要在其中安装示例 Memcached 工作负载和 CloudWatch 代理的、带有 EC2 启动类型的 Amazon ECS 集群，您可以通过输入以下命令来创建一个。

```
ecs-cli up --capability-iam --size 1 \
--instance-type t3.medium \
--cluster $ECS_CLUSTER_NAME \
--region $AWS_REGION
```

此命令的预期结果如下所示：

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker
 version 19.03.6-ce
INFO[0001] Created cluster                                cluster=ecs-ec2-memcached-
tutorial region=ca-central-1
INFO[0002] Waiting for your cluster resources to be created...
INFO[0002] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0063] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0124] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Security Group created: sg-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

安装 Memcached 示例工作负载

安装公开 Prometheus 指标的示例 Memcached 工作负载

1. 通过输入以下命令，下载 Memcached AWS CloudFormation 模板。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. 通过输入以下命令，设置要为 Memcached 创建的 IAM 角色名称。

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. 通过输入以下命令，安装示例 Memcached 工作负载。此示例在 `host` 网络模式下安装工作负载。

```
MEMCACHED_ECS_NETWORK_MODE=host

aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \
    --template-body file://memcached-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=ECSNetworkMode,ParameterValue=
$MEMCACHED_ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=
$MEMCACHED_ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$MEMCACHED_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

AWS CloudFormation 堆栈创建四种资源：

- 一个 ECS 任务角色
- 一个 ECS 任务执行角色
- 一个 Memcached 任务定义
- 一项 Memcached 服务

在 Memcached 任务定义中，定义了两个容器：

- 主容器运行一个简单的 Memcached 应用程序并打开端口 11211 以供访问。
- 另一个容器运行 Redis OSS 导出器进程，以在端口 9150 上公开 Prometheus 指标。该容器将由 CloudWatch 代理发现和抓取。

配置 CloudWatch 代理以抓取 Memcached Prometheus 指标

配置 CloudWatch 代理以抓取 Memcached Prometheus 指标

1. 通过输入以下命令，下载 `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` 的最新版本。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. 使用文本编辑器打开该文件，并在 `resource:CWAgentConfigSSMParameter` 部分中的 `value` 密钥后面查找完整的 CloudWatch 代理配置。

   然后，在 `ecs_service_discovery` 部分中，将以下配置添加到 `task_definition_list` 部分中。

```
{
    "sd_job_name": "ecs-memcached",
    "sd_metrics_ports": "9150",
    "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-demo.*:[0-9]+"
},
```

   对于 `metric_declaration` 部分，默认设置不允许任何 Memcached 指标。添加以下部分以允许 Memcached 指标。请务必遵循现有的缩进模式。

```
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
  "metric_selectors": [
    "^memcached_current_(bytes|items|connections)$",
    "^memcached_items_(reclaimed|evicted)_total$",
    "^memcached_(written|read)_bytes_total$",
```

```
    "^memcached_limit_bytes$",
    "^memcached_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily","status","command"],
 ["ClusterName", "TaskDefinitionFamily","command"]],
  "metric_selectors": [
    "^memcached_commands_total$"
  ]
},
```

3. 如果 AWS CloudFormation 已在 Amazon ECS 集群中部署 CloudWatch 代理，您可以通过输入以下命令来创建更改集。

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name memcached-scraping-support
```

4. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
5. 查看新创建的变更集 `memcached-scraping-support`。您会看到一项应用于 `CWAgentConfigSSMParameter` 资源的更改。通过输入以下命令执行变更集并重新启动 CloudWatch 代理任务。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
```

```
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. 等待大约 10 秒，然后输入以下命令。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. 如果您是首次为集群安装带有 Prometheus 指标收集功能的 CloudWatch 代理，请输入以下命令：

```
ECS_NETWORK_MODEE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

查看您的 Memcached 指标

本教程会将以下指标发送到 CloudWatch 中的 ECS/ContainerInsights/Prometheus 命名空间。您可以使用 CloudWatch 控制台查看该命名空间中的指标。

指标名称	尺寸	
memcached_current_items	ClusterName，TaskDefinitionFamily	

指标名称	尺寸	
memcached_current_connections	ClusterName ，TaskDefinitionFamily	
memcached_limit_bytes	ClusterName ，TaskDefinitionFamily	
memcached_current_bytes	ClusterName ，TaskDefinitionFamily	
memcached_written_bytes_total	ClusterName ，TaskDefinitionFamily	
memcached_read_bytes_total	ClusterName ，TaskDefinitionFamily	
memcached_items_evicted_total	ClusterName ，TaskDefinitionFamily	
memcached_items_reclaimed_total	ClusterName ，TaskDefinitionFamily	
memcached_commands_total	ClusterName ，TaskDefinitionFamily ClusterName 、TaskDefinitionFinitionFamy 、command ClusterName 、TaskDefinitionFamily、status、command	

> **Note**
>
> 命令维度的值可以是 `delete`、`get`、`cas`、`set`、`decr`、`touch`、`incr` 或 `flush`。
> 状态维度的值可以是 `hit`、`miss` 或 `badval`。

您还可以为 Memcached Prometheus 指标创建 CloudWatch 控制面板。

为 Memcached Prometheus 指标创建控制面板

1. 创建环境变量，替换以下值以匹配部署。

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-
$MEMCACHED_ECS_NETWORK_MOD
```

2. 输入以下命令以创建控制面板。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/ecs-task-definition-templates/deployment-mode/
replica-service/cwagent-prometheus/sample_cloudwatch_dashboards/memcached/
cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --region
 $AWS_REGION --dashboard-body
```

在 Amazon ECS Fargate 上抓取 Redis OSS Prometheus 指标的教程

本教程提供实践介绍，关于如何在 Amazon ECS Fargate 集群中抓取示例 Redis OSS 应用程序的 Prometheus 指标。Redis OSS Prometheus 导出器目标将由 CloudWatch 代理自动发现，并根据容器的 docker 标签提供 Prometheus 指标支持。

Redis OSS（https://redis.io/）是一个开源（获得 BSD 许可）的内存数据结构存储，用作数据库、缓存和消息代理。有关更多信息，请参阅 redis。

redis_exporter（获得 MIT 许可证）用于在指定端口（默认值：0.0.0.0:9121）上公开 Redis OSS prometheus 指标。有关更多信息，请参阅 redis_exporter。

本教程使用了以下两个 Docker Hub 存储库中的 Docker 镜像：

- redis
- redis_exporter

先决条件

要从 Amazon ECS 的示例 Prometheus 工作负载收集指标，您必须在集群中运行 Container Insights。有关安装 Container Insights 的信息，请参阅 在 Amazon ECS 上设置 Container Insights。

主题

- 设置 Amazon ECS Fargate 集群环境变量
- 为 Amazon ECS Fargate 集群设置网络环境变量
- 安装 Redis OSS 示例工作负载
- 配置 CloudWatch 代理以抓取 Redis OSS Prometheus 指标
- 查看您的 Redis OSS 指标

设置 Amazon ECS Fargate 集群环境变量

设置 Amazon ECS Fargate 集群环境变量

1. 请安装 Amazon ECS CLI（如果尚未安装）。有关更多信息，请参阅安装 Amazon ECS CLI。
2. 设置新的 Amazon ECS 集群名称和区域。例如：

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. （可选）如果您还没有要在其中安装示例 Redis OSS 工作负载和 CloudWatch 代理的 Amazon ECS Fargate 集群，可以通过输入以下命令来创建一个。

```
ecs-cli up --capability-iam \
--cluster $ECS_CLUSTER_NAME \
--launch-type FARGATE \
--region $AWS_DEFAULT_REGION
```

此命令的预期结果如下所示：

```
INFO[0000] Created cluster   cluster=ecs-fargate-redis-tutorial region=ca-central-1
INFO[0001] Waiting for your cluster resources to be created...
```

```
INFO[0001] Cloudformation stack status   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

为 Amazon ECS Fargate 集群设置网络环境变量

为 Amazon ECS Fargate 集群设置网络环境变量

1. 设置 Amazon ECS 集群的 VPC 和子网 ID。如果您在上一步骤中创建了新集群，您将在最终命令的结果中看到这些值。否则，请使用要与 Redis 一起使用的现有集群的 ID。

   ```
   ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxx
   ```

2. 在本教程中，我们将在 Amazon ECS 集群的 VPC 的默认安全组中安装 Redis OSS 应用程序和 CloudWatch 代理。默认安全组允许同一安全组内的所有网络连接，因此 CloudWatch 代理可以抓取 Redis OSS 容器上公开的 Prometheus 指标。在实际生产环境中，您可能希望为 Redis OSS 应用程序和 CloudWatch 代理创建专用安全组并为其设置自定义权限。

   要获取默认安全组 ID，请输入以下命令。

   ```
   aws ec2 describe-security-groups \
   --filters Name=vpc-id,Values=$ECS_CLUSTER_VPC  \
   --region $AWS_DEFAULT_REGION
   ```

   然后通过输入以下命令设置 Fargate 集群默认安全组变量，将 *my-default-security-group* 替换为您在上一个命令中找到的值。

   ```
   ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
   ```

安装 Redis OSS 示例工作负载

安装公开 Prometheus 指标的示例 Redis OSS 工作负载

1. 通过输入以下命令，下载 Redis OSS AWS CloudFormation 模板。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. 通过输入以下命令，设置要为 Redis OSS 创建的 IAM 角色名称。

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. 通过输入以下命令安装 Redis OSS 示例工作负载。

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-fargate-awsvpc \
    --template-body file://redis-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \
                 ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME
 \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$REDIS_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_DEFAULT_REGION
```

AWS CloudFormation 堆栈创建四种资源：

- 一个 ECS 任务角色
- 一个 ECS 任务执行角色
- 一个 Redis OSS 任务定义
- 一项 Redis OSS 服务

在 Redis OSS 任务定义中，定义了两个容器：

- 主容器运行一个简单的 Redis OSS 应用程序并打开端口 6379 以供访问。
- 另一个容器运行 Redis OSS 导出器进程，以在端口 9121 上公开 Prometheus 指标。该容器将由 CloudWatch 代理发现和抓取。定义了以下 docker 标签，以便 CloudWatch 代理可以根据它发现该容器。

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

配置 CloudWatch 代理以抓取 Redis OSS Prometheus 指标

配置 CloudWatch 代理以抓取 Redis OSS Prometheus 指标

1. 通过输入以下命令，下载 `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` 的最新版本。

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
   ```

2. 使用文本编辑器打开该文件，并在 `resource:CWAgentConfigSSMParameter` 部分中的 `value` 密钥后面查找完整的 CloudWatch 代理配置。

   然后，在此处显示的 `ecs_service_discovery` 部分中，使用基于 `ECS_PROMETHEUS_EXPORTER_PORT` 的默认设置启用基于 `docker_label` 的服务发现，这与我们在 Redis OSS ECS 任务定义中定义的 Docker 标签相匹配。因此，我们无需在此部分进行任何更改：

   ```
   ecs_service_discovery": {
     "sd_frequency": "1m",
     "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
   *  "docker_label": {
     },*
     ...
   ```

   对于 `metric_declaration` 部分，默认设置不允许任何 Redis OSS 指标。添加以下部分以允许 Redis OSS 指标。请务必遵循现有的缩进模式。

   ```
   {
     "source_labels": ["container_name"],
     "label_matcher": "^redis-exporter-.*$",
     "dimensions": [["ClusterName","TaskDefinitionFamily"]],
     "metric_selectors": [
       "^redis_net_(in|out)put_bytes_total$",
       "^redis_(expired|evicted)_keys_total$",
   ```

```
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

3. 如果 AWS CloudFormation 已在 Amazon ECS 集群中部署 CloudWatch 代理，您可以通过输入以下命令来创建更改集。

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
```

```
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --change-set-name redis-scraping-support
```

4. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
5. 查看新创建的变更集 `redis-scraping-support`。您会看到一项应用于 `CWAgentConfigSSMParameter` 资源的更改。通过输入以下命令执行变更集并重新启动 CloudWatch 代理任务。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

6. 等待大约 10 秒，然后输入以下命令。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

7. 如果您是首次为集群安装带有 Prometheus 指标收集功能的 CloudWatch 代理，请输入以下命令：

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
```

```
    --region ${AWS_DEFAULT_REGION}
```

查看您的 Redis OSS 指标

本教程会将以下指标发送到 CloudWatch 中的 ECS/ContainerInsights/Prometheus 命名空间。您可以使用 CloudWatch 控制台查看该命名空间中的指标。

指标名称	尺寸	
redis_net_input_bytes_total	ClusterName、TaskDefinitionFamily	
redis_net_output_bytes_total	ClusterName、TaskDefinitionFamily	
redis_expired_keys_total	ClusterName、TaskDefinitionFamily	
redis_evicted_keys_total	ClusterName、TaskDefinitionFamily	
redis_keyspace_hits_total	ClusterName、TaskDefinitionFamily	
redis_keyspace_misses_total	ClusterName、TaskDefinitionFamily	
redis_memory_used_bytes	ClusterName、TaskDefinitionFamily	
redis_connected_clients	ClusterName、TaskDefinitionFamily	

指标名称	尺寸	
redis_commands_total	ClusterName，TaskDefinitionFamily，cmd	
redis_db_keys	ClusterName，TaskDefinitionFamily，db	

> **Note**
>
> cmd 维度的值可以是 append、client、command、config、dbsize、flushall、get、incr、info、latency 或 slowlog。
> db 维度的值可以从 db0 到 db15。

您还可以为 Redis OSS Prometheus 指标创建 CloudWatch 控制面板。

为 Redis OSS Prometheus 指标创建控制面板

1. 创建环境变量，替换以下值以匹配部署。

```
DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo-$ECS_CLUSTER_NAME-fargate-awsvpc
```

2. 输入以下命令以创建控制面板。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

在 Amazon EKS 和 Kubernetes 集群上设置和配置 Prometheus 指标收集

要从运行 Amazon EKS 或 Kubernetes 的集群收集 Prometheus 指标，您可以使用 CloudWatch 代理作为收集器或使用 AWS Distro for OpenTelemetry 收集器。有关使用 AWS Distro for OpenTelemetry

收集器的信息，请参阅 https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus。

下面几节介绍如何使用 CloudWatch 代理收集 Prometheus 指标。他们说明了如何在运行 Amazon EKS 或 Kubernetes 的集群上安装带有 Prometheus 监控功能的 CloudWatch 代理，以及如何配置该代理以抓取其他目标。他们还提供用于设置示例工作负载的可选教程，以使用 Prometheus 监控进行测试。

主题

- 在 Amazon EKS 和 Kubernetes 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理

在 Amazon EKS 和 Kubernetes 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理

本节介绍如何在运行 Amazon EKS 或 Kubernetes 的集群中设置带有 Prometheus 监控功能的 CloudWatch 代理。执行此操作后，代理会自动抓取并导入该集群中运行的以下工作负载的指标。

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

您还可以将代理配置为抓取和导入其他 Prometheus 工作负载和源。

在按照下面这些步骤安装 CloudWatch 代理来收集 Prometheus 指标之前，您必须有在 Amazon EKS 上运行的集群或者在 Amazon EC2 实例上运行的 Kubernetes 集群。

VPC 安全组要求

Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。

CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

主题

- 在 Amazon EKS 和 Kubernetes 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理
- 抓取其他 Prometheus 源并导入这些指标
- （可选）为 Prometheus 指标测试设置示例容器化 Amazon EKS 工作负载

在 Amazon EKS 和 Kubernetes 集群上安装带有 Prometheus 指标收集功能的 CloudWatch 代理

本节介绍如何在运行 Amazon EKS 或 Kubernetes 的集群中设置带有 Prometheus 监控功能的 CloudWatch 代理。执行此操作后，代理会自动抓取并导入该集群中运行的以下工作负载的指标。

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

您还可以将代理配置为抓取和导入其他 Prometheus 工作负载和源。

在按照下面这些步骤安装 CloudWatch 代理来收集 Prometheus 指标之前，您必须有在 Amazon EKS 上运行的集群或者在 Amazon EC2 实例上运行的 Kubernetes 集群。

VPC 安全组要求

Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。

CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

主题

- 设置 IAM 角色
- 安装 CloudWatch 代理以收集 Prometheus 指标

设置 IAM 角色

第一步是在集群中设置必要的 IAM 角色。有两种方法：

- 为服务账户设置 IAM 角色，也称为服务角色。此方法适用于 EC2 启动类型和 Fargate 启动类型。
- 将 IAM 策略添加到集群的 IAM 角色。这仅适用于 EC2 启动类型。

设置服务角色（EC2 启动类型和 Fargate 启动类型）

要设置服务角色，请输入以下命令。将 *MyCluster* 替换为集群的名称。

```
eksctl create iamserviceaccount \
 --name cwagent-prometheus \
--namespace amazon-cloudwatch \
 --cluster MyCluster \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

向集群的 IAM 角色添加策略（仅限 EC2 启动类型）

在集群中为 Prometheus 支持设置 IAM 策略

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择实例。
3. 您需要查找集群的 IAM 角色名称的前缀。为此，请选中集群中实例名称旁边的复选框，然后选择操作、实例设置、附加/替换 IAM 角色。然后复制 IAM 角色的前缀，例如 `eksctl-dev303-workshop-nodegroup`。
4. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
5. 在导航窗格中，选择角色。
6. 使用搜索框查找您在此过程前面复制的前缀，然后选择该角色。
7. 选择附加策略。
8. 使用搜索框查找 CloudWatchAgentServerPolicy。选中 CloudWatchAgentServerPolicy 旁边的复选框，然后选择附加策略。

安装 CloudWatch 代理以收集 Prometheus 指标

您必须在集群中安装 CloudWatch 代理才能收集指标。对于 Amazon EKS 集群和 Kubernetes 集群，安装代理的方式有所不同。

删除具有 Prometheus 支持的 CloudWatch 代理的早期版本

如果您的集群中已安装了具有 Prometheus 支持的 CloudWatch 代理版本，则必须通过输入以下命令删除该版本。这仅对具有 Prometheus 支持的以前版本的代理是必要的。您无需删除启用了 Container Insights 但没有 Prometheus 支持的 CloudWatch 代理。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

在具有 EC2 启动类型的 Amazon EKS 集群上安装 CloudWatch 代理

要在 Amazon EKS 集群上安装具有 Prometheus 支持的 CloudWatch 代理，请按照下列步骤操作。

在 Amazon EKS 集群上安装具有 Prometheus 支持的 CloudWatch 代理

1. 输入以下命令，检查是否已创建 amazon-cloudwatch 命名空间：

   ```
   kubectl get namespace
   ```

2. 如果结果中未显示 amazon-cloudwatch，请输入以下命令来创建它：

   ```
   kubectl create namespace amazon-cloudwatch
   ```

3. 要使用默认配置部署代理并使其将数据发送到所在的 AWS 区域，请输入以下命令：

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
   container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
   service/cwagent-prometheus/prometheus-eks.yaml
   ```

   要让代理将数据发送到不同的区域，请按照下列步骤操作：

   a. 输入以下命令，下载代理的 YAML 文件：

      ```
      curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
      container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
      service/cwagent-prometheus/prometheus-eks.yaml
      ```

   b. 使用文本编辑器打开文件，然后搜索文件的 cwagentconfig.json 块。

   c. 添加突出显示的行，并指定所需的区域：

      ```
      cwagentconfig.json: |
          {
            "agent": {
              "region": "us-east-2"
      ```

```
      },
      "logs": { ...
```

d. 保存文件并使用更新后的文件部署代理。

```
kubectl apply -f prometheus-eks.yaml
```

在具有 Fargate 启动类型的 Amazon EKS 集群上安装 CloudWatch 代理

要在具有 Fargate 启动类型的 Amazon EKS 集群上安装具有 Prometheus 支持的 CloudWatch 代理，请按照下列步骤操作。

在具有 Fargate 启动类型的 Amazon EKS 集群上安装具有 Prometheus 支持的 CloudWatch 代理

1. 输入以下命令为 CloudWatch 代理创建 Fargate 配置文件，以便它可以在集群内运行。将 *MyCluster* 替换为集群的名称。

```
eksctl create fargateprofile --cluster MyCluster \
--name amazon-cloudwatch \
--namespace amazon-cloudwatch
```

2. 要安装 CloudWatch 代理，请输入以下命令。将 *MyCluster* 替换为集群的名称。此名称在存储代理收集的日志事件的日志组名称中使用，也用作代理收集的指标的维度。

   将 *region####*替换为要将指标发送到的区域的名称。例如，us-west-1。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-eks-fargate.yaml |
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
kubectl apply -f -
```

在 Kubernetes 集群上安装 CloudWatch 代理

要在运行 Kubernetes 的集群上安装具有 Prometheus 支持的 CloudWatch 代理，请输入以下命令：

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-k8s.yaml |
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
```

```
kubectl apply -f -
```

将 *MyCluster* 替换为集群的名称。此名称在存储代理收集的日志事件的日志组名称中使用，也用作代理收集的指标的维度。

将 *region####*替换为要将指标发送到的 AWS 区域的名称。例如，**us-west-1**。

验证代理是否正在运行

在 Amazon EKS 和 Kubernetes 集群上，您可以输入以下命令以确认代理正在运行。

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

如果结果包含处于 Running 状态的单个 CloudWatch 代理 pod，则代理会运行并收集 Prometheus 指标。默认情况下，CloudWatch 代理每分钟会收集 App Mesh、NGINX、Memcached、Java/JMX 和 HAProxy 的指标。有关这些指标的更多信息，请参阅 CloudWatch 代理收集的 Prometheus 指标。有关如何在 CloudWatch 中查看 Prometheus 指标的说明，请参阅 查看 Prometheus 指标

您还可以将 CloudWatch 代理配置为从其他 Prometheus 导出程序收集指标。有关更多信息，请参阅抓取其他 Prometheus 源并导入这些指标。

抓取其他 Prometheus 源并导入这些指标

具有 Prometheus 监控功能的 CloudWatch 代理需要两种配置来抓取 Prometheus 指标。一个用于 Prometheus 文档中 <scrape_config> 记录的标准 Prometheus 配置。另一种配置是 CloudWatch 代理配置文件。

对于 Amazon EKS 集群，配置在 `prometheus-eks.yaml`（适用于 EC2 启动类型）或 `prometheus-eks-fargate.yaml`（适用于 Fargate 启动类型）中定义为两个 Config 映射：

- `name: prometheus-config` 部分包含 Prometheus 抓取的设置。
- `name: prometheus-cwagentconfig` 部分包含 CloudWatch 代理的配置。您可以使用此部分配置 CloudWatch 如何收集 Prometheus 指标。例如，您指定要导入 CloudWatch 的指标，并定义其维度。

对于在 Amazon EC2 实例上运行的 Kubernetes 集群，配置在 `prometheus-k8s.yaml` YAML 文件中定义为两个 config 映射：

- `name: prometheus-config` 部分包含 Prometheus 抓取的设置。
- `name: prometheus-cwagentconfig` 部分包含 CloudWatch 代理的配置。

要抓取其他 Prometheus 指标源并将这些指标导入 CloudWatch，您需要修改 Prometheus 抓取配置和 CloudWatch 代理配置，然后使用更新的配置重新部署代理。

VPC 安全组要求

Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。

CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

Prometheus 抓取配置

CloudWatch 代理支持标准的 Prometheus 抓取配置，如 Prometheus 文档中的 <scrape_config> 所述。您可以编辑此部分以更新此文件中已有的配置，并添加其他 Prometheus 抓取目标。默认情况下，示例配置文件包含以下全局配置行：

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- scrape_interval – 定义抓取目标的频率。
- scrape_timeout – 定义抓取请求超时之前的等待时间。

您还可以在作业级别为这些设置定义不同的值，以覆盖全局配置。

Prometheus 抓取任务

CloudWatch 代理 YAML 文件已配置了一些默认的抓取任务。例如在 `prometheus-eks.yaml` 中，在 `scrape_configs` 部分中的 `job_name` 行中配置了默认的抓取任务。在此文件中，以下默认 `kubernetes-pod-jmx` 部分会抓取 JMX Exporter 指标。

```
   - job_name: 'kubernetes-pod-jmx'
      sample_limit: 10000
      metrics_path: /metrics
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__address__]
        action: keep
```

```
        regex: '.*:9404$'
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: pod_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_container_name
        target_label: container_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_controller_name
        target_label: pod_controller_name
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_controller_kind
        target_label: pod_controller_kind
      - action: replace
        source_labels:
        - __meta_kubernetes_pod_phase
        target_label: pod_phase
```

这些默认目标中的每个目标都会被抓取，并使用嵌入式指标格式在日志事件中将指标发送到 CloudWatch。有关更多信息，请参阅 在日志中嵌入指标。

来自 Amazon EKS 和 Kubernetes 集群的日志事件存储在 CloudWatch Logs 的 /aws/containerinsights/***cluster_name***/prometheus 日志组中。来自 Amazon ECS 集群的日志事件存储在 /aws/ecs/containerinsights/***cluster_name***/prometheus 日志组中。

每个抓取作业都包含在此日志组中的不同日志流中。例如，为 App Mesh 定义了 Prometheus 抓取任务 `kubernetes-pod-appmesh-envoy`。所有来自 Amazon EKS 和 Kubernetes 集群的 App Mesh Prometheus 指标都发送到名为 /aws/containerinsights/***cluster_name***>prometheus/kubernetes-pod-appmesh-envoy/ 的日志流。

要添加新的抓取目标，请将新的 `job_name` 部分添加到 YAML 文件的 `scrape_configs` 部分，然后重新启动代理。有关此过程的示例，请参阅 添加新 Prometheus 抓取目标的教程：Prometheus API 服务器指标。

Prometheus 的 CloudWatch 代理配置

CloudWatch 代理配置文件在 `metrics_collected` 下面有一个 `prometheus` 部分用于 Prometheus 抓取配置。它包含以下配置选项：

- cluster_name – 指定要在日志事件中添加为标签的集群名称。该字段是可选的。如果省略它，代理可以检测到 Amazon EKS 或 Kubernetes 集群名称。
- log_group_name – 指定已抓取 Prometheus 指标的日志组名称。该字段是可选的。如果省略它，CloudWatch 会将 /aws/containerinsights/***cluster_name***/prometheus 用于来自 Amazon EKS 和 Kubernetes 集群的日志。
- prometheus_config_path – 指定 Prometheus 抓取配置文件路径。如果此字段的值以 `env:` 开头，Prometheus 抓取配置文件内容将从容器的环境变量中检索。请勿更改此字段。
- ecs_service_discovery – 是指定 Amazon ECS Prometheus 服务发现配置的部分。有关更多信息，请参阅 Amazon ECS 集群上自动发现的详细指南。

  `ecs_service_discovery` 部分包含以下字段：

  - `sd_frequency` 是发现 Prometheus 导出器的频率。指定数字和单位后缀。例如，`1m` 表示每分钟一次或 `30s` 表示每 30 秒一次。有效的单位后缀为 `ns`、`us`、`ms`、`s`、`m` 和 `h`。

    该字段是可选的。默认值为 60 秒（1 分钟）。
  - `sd_target_cluster` 是用于自动发现的目标 Amazon ECS 集群名称。该字段是可选的。默认名称为安装 CloudWatch 代理的 Amazon ECS 集群的名称。
  - `sd_cluster_region` 是目标 Amazon ECS 集群的区域。该字段是可选的。默认区域为安装 CloudWatch 代理的 Amazon ECS 集群的区域。
  - `sd_result_file` 是 Prometheus 目标结果的 YAML 文件的路径。Prometheus 抓取配置将引用此文件。
  - `docker_label` 是可选部分，您可以使用它来指定基于 docker 标签的服务发现的配置。如果省略此部分，则不会使用基于 docker 标签的发现。此部分包含以下字段：
    - `sd_port_label` 是容器的 docker 标签名称，用于指定 Prometheus 指标的容器端口。默认值为 `ECS_PROMETHEUS_EXPORTER_PORT`。如果容器没有此 docker 标签，CloudWatch 代理将跳过它。
    - `sd_metrics_path_label` 是容器的 docker 标签名称，用于指定 Prometheus 指标路径。默认值为 `ECS_PROMETHEUS_METRICS_PATH`。如果容器没有此 docker 标签，则代理会采用默认路径 `/metrics`。

    - sd_job_name_label 是容器的 docker 标签名称，用于指定 Prometheus 抓取任务名称。默认值为 job。如果容器没有此 docker 标签，CloudWatch 代理会使用 Prometheus 抓取配置中的任务名称。
  - task_definition_list 是可选部分，可用于指定基于任务定义的服务发现的配置。如果省略此部分，则不会使用基于任务定义的发现。此部分包含以下字段：
    - sd_task_definition_arn_pattern 是用于指定要发现的 Amazon ECS 任务定义的模式。这是正则表达式。
    - sd_metrics_ports 列出 Prometheus 指标的 containerPort。用分号分隔 containerPorts。
    - sd_container_name_pattern 指定 Amazon ECS 任务容器名称。这是正则表达式。
    - sd_metrics_path 指定 Prometheus 指标路径。如果省略此项，代理会采用默认路径 /metrics
    - sd_job_name 指定 Prometheus 抓取任务名称。如果省略此字段，CloudWatch 代理会使用 Prometheus 抓取配置中的任务名称。
- metric_declaration – 是指定要生成的采用嵌入式指标格式的日志数组的部分。默认情况下，CloudWatch 代理从中进行导入的每个 Prometheus 源都有 metric_declaration 部分。这些部分各包括以下字段：
  - label_matcher 是一个正则表达式，用于检查 source_labels 中列出的标签的值。匹配的指标将启用，以包含在发送到 CloudWatch 的嵌入式指标格式中。

    如果您在 source_labels 中指定了多个标签，我们建议您不要在 label_matcher 的正则表达式中使用 ^ 或 $ 字符。
  - source_labels 指定由 label_matcher 行检查的标签的值。
  - label_separator 指定要在 label_matcher 行中使用的分隔符（如果指定了多个 source_labels）。默认为 ;。您可以在以下示例中看到 label_matcher 行中使用的此默认值。
  - metric_selectors 是一个正则表达式，用于指定要收集并发送到 CloudWatch 的指标。
  - dimensions 是要用作每个选定指标的 CloudWatch 维度的标签列表。

请参阅以下 metric_declaration 示例。

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system",
     "dimensions":[
```

```
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

此示例配置嵌入式指标格式部分，以便在满足以下条件时作为日志事件发送：

- Service 的值包含 node-exporter 或 kube-dns。
- Namespace 的值为 kube-system。
- Prometheus 指标 coredns_dns_request_type_count_total 同时包含 Service 和 Namespace 标签。

发送的日志事件包括以下突出显示的部分：

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
   "Service":"kube-dns",
   "coredns_dns_request_type_count_total":2562,
   "eks_amazonaws_com_component":"kube-dns",
   "instance":"192.168.61.254:9153",
   "job":"kubernetes-service-endpoints",
   ...
```

```
}
```

添加新 Prometheus 抓取目标的教程：Prometheus API 服务器指标

默认情况下，Kubernetes API 服务器会在端点上公开 Prometheus 指标。Kubernetes API 服务器抓取配置的官方示例可在 Github 上找到。

以下教程演示如何执行以下步骤以开始将 Kubernetes API 服务器指标导入到 CloudWatch 中：

- 将 Kubernetes API 服务器的 Prometheus 抓取配置添加到 CloudWatch 代理 YAML 文件。
- 在 CloudWatch 代理 YAML 文件中配置嵌入式指标格式指标定义。
- （可选）为 Kubernetes API 服务器指标创建 CloudWatch 控制面板。

> **Note**
>
> Kubernetes API 服务器公开计量表、计数器、直方图和摘要指标。在此版本的 Prometheus 指标支持中，CloudWatch 仅导入具有计量表、计数器和汇总类型的指标。

开始在 CloudWatch 中收集 Kubernetes API 服务器 Prometheus 指标

1. 通过输入以下命令之一，下载 `prometheus-eks.yaml`、`prometheus-eks-fargate.yaml` 或 `prometheus-k8s.yaml` 文件的最新版本。

   对于具有 EC2 启动类型的 Amazon EKS 集群，请输入以下命令：

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   对于具有 Fargate 启动类型的 Amazon EKS 集群，请输入以下命令：

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
   ```

   对于在 Amazon EC2 实例上运行的 Kubernetes 集群，请输入以下命令：

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-k8s.yaml
```

2. 使用文本编辑器打开文件，找到 prometheus-config 部分，然后在该部分中添加以下部分。然后，保存更改：

```
    # Scrape config for API servers
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
        - role: endpoints
          namespaces:
            names:
              - default
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name]
        action: keep
        regex: kubernetes;https
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - action: replace
        source_labels:
        - __meta_kubernetes_service_name
        target_label: Service
```

3. 当 YAML 文件仍在文本编辑器中处于打开状态时，找到 cwagentconfig.json 部分。添加以下子部分并保存更改。此部分将 API 服务器指标放到 CloudWatch 代理允许列表中。将三种类型的 API 服务器指标添加到允许列表中：

   - etcd 对象计数
   - API 服务器注册控制器指标
   - API 服务器请求指标

```
{"source_labels": ["job", "resource"],
  "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|
deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",
  "dimensions": [["ClusterName","Service","resource"]],
  "metric_selectors": [
  "^etcd_object_counts$"
  ]
},
{"source_labels": ["job", "name"],
   "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",
   "dimensions": [["ClusterName","Service","name"]],
   "metric_selectors": [
   "^workqueue_depth$",
   "^workqueue_adds_total$",
   "^workqueue_retries_total$"
  ]
},
{"source_labels": ["job","code"],
  "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",
  "dimensions": [["ClusterName","Service","code"]],
  "metric_selectors": [
   "^apiserver_request_total$"
  ]
},
{"source_labels": ["job"],
  "label_matcher": "^kubernetes-apiservers",
  "dimensions": [["ClusterName","Service"]],
  "metric_selectors": [
  "^apiserver_request_total$"
  ]
},
```

4. 如果您已在集群中部署了具有 Prometheus 支持的 CloudWatch 代理，则必须通过输入以下命令将其删除：

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

5. 通过输入以下任一命令，使用更新的配置部署 CloudWatch 代理。对于具有 EC2 启动类型的 Amazon EKS 集群，请输入：

```
kubectl apply -f prometheus-eks.yaml
```

对于具有 Fargate 启动类型的 Amazon EKS 集群，请输入以下命令。将 *MyCluster* 和 *region####*替换为值以匹配您的部署。

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

对于 Kubernetes 集群，请输入以下命令。将 *MyCluster* 和 *region####*替换为值以匹配您的部署。

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

完成此操作后，您应该会看到 /aws/containerinsights/***cluster_name***/prometheus 日志组中名为 kubernetes-apiservers 的新日志流。此日志流应使用嵌入式指标格式定义来嵌入日志事件，如下所示：

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"apiserver_request_total"
            }
         ],
         "Dimensions":[
            [
               "ClusterName",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "ClusterName":"my-cluster-name",
   "Namespace":"default",
```

```
    "Service":"kubernetes",
    "Timestamp":"1592267020339",
    "Version":"0",
    "apiserver_request_count":0,
    "apiserver_request_total":0,
    "code":"0",
    "component":"apiserver",
    "contentType":"application/json",
    "instance":"192.0.2.0:443",
    "job":"kubernetes-apiservers",
    "prom_metric_type":"counter",
    "resource":"pods",
    "scope":"namespace",
    "verb":"WATCH",
    "version":"v1"
}
```

您可以在 ContainerInsights/Prometheus 命名空间中的 CloudWatch 控制台中查看您的指标。（可选）还可以为 Prometheus Kubernetes API 服务器指标创建 CloudWatch 控制面板。

（可选）为 Kubernetes API 服务器指标创建控制面板

要在控制面板中查看 Kubernetes API 服务器指标，您必须先完成前面几个部分中的步骤，然后才能开始在 CloudWatch 中收集这些指标。

为 Kubernetes API 服务器指标创建控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 确保选择了正确的 AWS 区域。
3. 在导航窗格中，选择控制面板。
4. 选择创建控制面板。输入新控制面板的名称，然后选择创建控制面板。
5. 在添加到该控制面板中，选择取消。
6. 依次选择操作、编辑控制面板。
7. 下载以下 JSON 文件：Kubernetes API 控制面板源。
8. 使用文本编辑器打开下载的 JSON 文件，然后执行以下更改：
   - 将所有 {{YOUR_CLUSTER_NAME}} 字符串替换为您的集群的确切名称。请勿在文本之前或之后添加空格。

- 将所有 {{YOUR_AWS_REGION}} 字符串替换为在其中收集指标的区域的名称。例如 us-west-2。请勿在文本之前或之后添加空格。

9. 复制整个 JSON blob 并将其粘贴到 CloudWatch 控制台的文本框中，替换框中已有的内容。
10. 选择更新、保存控制面板。

（可选）为 Prometheus 指标测试设置示例容器化 Amazon EKS 工作负载

要测试 CloudWatch Container Insights 中的 Prometheus 指标支持，您可设置以下一个或多个容器化工作负载。具有 Prometheus 支持的 CloudWatch 代理会自动从这些工作负载中的每一个收集指标。要查看默认情况下收集的指标，请参阅 CloudWatch 代理收集的 Prometheus 指标。

在安装任意这些工作负载之前，您必须输入以下命令来安装 Helm 3.x：

```
brew install helm
```

有关更多信息，请参阅 Helm。

主题

- 为 Amazon EKS 和 Kubernetes 设置 AWS App Mesh 示例工作负载
- 使用 Amazon EKS 和 Kubernetes 上的示例流量设置 NGINX
- 使用 Amazon EKS 和 Kubernetes 上的指标导出器设置 memcached
- 在 Amazon EKS 和 Kubernetes 上设置 Java/JMX 示例工作负载
- 使用 Amazon EKS 和 Kubernetes 上的指标导出器设置 HAProxy
- 添加新 Prometheus 抓取目标的教程：Amazon EKS 和 Kubernetes 集群上的 Redis OSS

为 Amazon EKS 和 Kubernetes 设置 AWS App Mesh 示例工作负载

CloudWatch Container Insights 支持 AWS App Mesh 中的 Prometheus 支持。以下部分介绍了如何设置 App Mesh。

CloudWatch Container Insights 还可以收集 App Mesh Envoy 访问日志。有关更多信息，请参阅（可选）启用 App Mesh Envoy 访问日志。

主题

- 在具有 EC2 启动类型或 Kubernetes 集群的 Amazon EKS 集群上设置 AWS App Mesh 示例工作负载

- 使用 Fargate 启动类型在 Amazon EKS 集群上设置 AWS App Mesh 示例工作负载

在具有 EC2 启动类型或 Kubernetes 集群的 Amazon EKS 集群上设置 AWS App Mesh 示例工作负载

如果您要在运行 Amazon EKS 且具有 EC2 启动类型的集群或 Kubernetes 集群上设置 App Mesh，请使用这些说明。

配置 IAM 权限

必须将 AWSAppMeshFullAccess 策略添加到您的 Amazon EKS 或 Kubernetes 节点组的 IAM 角色中。在 Amazon EKS 上，此节点组名称类似于 `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL`。在 Kubernetes 上，它可能类似于 `nodes.integ-test-kops-prometheus.k8s.local`。

安装 App Mesh

要安装 App Mesh Kubernetes 控制器，请按照 App Mesh 控制器中的说明进行操作。

安装示例应用程序

aws-app-mesh-examples 包含多个 Kubernetes App Mesh 演练。在本教程中，您将安装一个示例颜色应用程序，该应用程序会展示 http 路由如何使用标头来匹配传入请求。

使用示例 App Mesh 应用程序来测试 Container Insights

1. 按照以下说明安装应用程序：https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-http-headers。
2. 启动 curler pod 以生成流量：

```
kubectl -n default run -it curler --image=tutum/curl /bin/bash
```

3. 通过更改 HTTP 标头来对不同的端点执行 curl。多次运行 curl 命令，如下所示：

```
curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/;
 echo;

curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/;
 echo;

curl -H "color_header: yellow" front.howto-k8s-http-
headers.svc.cluster.local:8080/; echo;
```

4. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
5. 在集群运行的 AWS 区域的导航窗格中，选择 Metrics（指标）。指标位于 ContainerInsights/Prometheus 命名空间中。
6. 要查看 CloudWatch Logs 事件，请在导航窗格中选择 Log groups（日志组）。事件位于日志流 `kubernetes-pod-appmesh-envoy` 的日志组 `/aws/containerinsights/your_cluster_name/prometheus` 中。

删除 App Mesh 测试环境

App Mesh 和示例应用程序使用完成后，请使用以下命令删除不必要的资源。输入以下命令删除示例应用程序：

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/
kubectl delete -f _output/manifest.yaml
```

输入以下命令删除 App Mesh 控制器：

```
helm delete appmesh-controller -n appmesh-system
```

使用 Fargate 启动类型在 Amazon EKS 集群上设置 AWS App Mesh 示例工作负载

如果您要在运行 Amazon EKS 且具有 Fargate 启动类型的集群上设置 App Mesh，请遵循这些说明。

配置 IAM 权限

要设置 IAM 权限，请输入以下命令。将 *MyCluster* 替换为您的集群的名称。

```
eksctl create iamserviceaccount --cluster MyCluster \
 --namespace howto-k8s-fargate \
 --name appmesh-pod \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
 --override-existing-serviceaccounts \
 --approve
```

安装 App Mesh

要安装 App Mesh Kubernetes 控制器，请按照 App Mesh 控制器中的说明进行操作。请务必按照有关 Fargate 启动类型的 Amazon EKS 的说明进行操作。

安装示例应用程序

aws-app-mesh-examples 包含多个 Kubernetes App Mesh 演练。在本教程中，您将安装适用于 Fargate 启动类型的 Amazon EKS 集群的示例颜色应用程序。

使用示例 App Mesh 应用程序来测试 Container Insights

1. 按照以下说明安装应用程序：https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-fargate。

   这些说明假定您正在使用正确的 Fargate 配置文件创建新集群。如果您想使用已设置的 Amazon EKS 集群，您可以使用以下命令为本演示设置该集群。将 *MyCluster* 替换为您的集群的名称。

   ```
   eksctl create iamserviceaccount --cluster MyCluster \
    --namespace howto-k8s-fargate \
    --name appmesh-pod \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
    --override-existing-serviceaccounts \
    --approve
   ```

   ```
   eksctl create fargateprofile --cluster MyCluster \
   --namespace howto-k8s-fargate --name howto-k8s-fargate
   ```

2. 端口转发前端应用程序部署：

   ```
   kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
   ```

3. Curl 前端应用程序：

   ```
   while true; do  curl -s http://localhost:8080/color; sleep 0.1; echo ; done
   ```

4. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

5. 在集群运行的 AWS 区域的导航窗格中，选择 Metrics（指标）。指标位于 ContainerInsights/Prometheus 命名空间中。
6. 要查看 CloudWatch Logs 事件，请在导航窗格中选择 Log groups（日志组）。事件位于日志流 `kubernetes-pod-appmesh-envoy` 的日志组 `/aws/containerinsights/your_cluster_name/prometheus` 中。

删除 App Mesh 测试环境

App Mesh 和示例应用程序使用完成后，请使用以下命令删除不必要的资源。输入以下命令删除示例应用程序：

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/
kubectl delete -f _output/manifest.yaml
```

输入以下命令删除 App Mesh 控制器：

```
helm delete appmesh-controller -n appmesh-system
```

使用 Amazon EKS 和 Kubernetes 上的示例流量设置 NGINX

NGINX 是一个 Web 服务器，也可以用作负载均衡器和反向代理。有关 Kubernetes 如何使用 NGINX for ingress 的更多信息，请参阅 kubernetes/ingress-nginx。

安装带有示例流量服务的 Ingress-NGINX 以测试 Container Insights Prometheus 支持

1. 输入以下命令以添加 Helm ingress-nginx 存储库：

   ```
   helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
   ```

2. 输入以下命令：

   ```
   kubectl create namespace nginx-ingress-sample

   helm install my-nginx ingress-nginx/ingress-nginx \
   --namespace nginx-ingress-sample \
   --set controller.metrics.enabled=true \
   --set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
   --set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
   ```

3. 通过输入以下命令检查服务是否正确启动：

```
kubectl get service -n nginx-ingress-sample
```

此命令的输出应显示多列，包括一个 EXTERNAL-IP 列。

4. 将 EXTERNAL-IP 变量设置为 NGINX 摄取控制器的行中 EXTERNAL-IP 列的值。

```
EXTERNAL_IP=your-nginx-controller-external-ip
```

5. 输入以下命令启动一些示例 NGINX 流量。

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. 输入以下命令以确认所有三个 pod 都处于 Running 状态。

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

如果它们正在运行，您应该很快在 ContainerInsights/Prometheus 命名空间中看到指标。

卸载 NGINX 和示例流量应用程序

1. 输入以下命令删除示例流量服务：

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. 按 Helm 版本名称删除 NGINX 出口。

```
helm uninstall my-nginx --namespace nginx-ingress-sample
kubectl delete namespace nginx-ingress-sample
```

使用 Amazon EKS 和 Kubernetes 上的指标导出器设置 memcached

memcached 是一个开源内存对象缓存系统。有关更多信息，请参阅什么是 Memcached？

如果您在具有 Fargate 启动类型的集群上运行 memcached，则需要在执行此过程中的步骤之前设置 Fargate 配置文件。要设置配置文件，请输入以下命令。将 *MyCluster* 替换为您的集群的名称。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace memcached-sample --name memcached-sample
```

安装带有 Metric Exporter 的 memcached 以测试 Container Insights Prometheus 支持

1. 输入以下命令以添加存储库：

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami
   ```

2. 输入以下命令以创建新的命名空间：

   ```
   kubectl create namespace memcached-sample
   ```

3. 输入以下命令以安装 Memcached

   ```
   helm install my-memcached bitnami/memcached --namespace memcached-sample \
   --set metrics.enabled=true \
   --set-string serviceAnnotations.prometheus\\.io/port="9150" \
   --set-string serviceAnnotations.prometheus\\.io/scrape="true"
   ```

4. 输入以下命令以确认正在运行的服务的注释：

   ```
   kubectl describe service my-memcached-metrics -n memcached-sample
   ```

   您应该看到以下两个注释：

   ```
   Annotations:   prometheus.io/port: 9150
                  prometheus.io/scrape: true
   ```

卸载 memcached

- 输入以下命令：

  ```
  helm uninstall my-memcached --namespace memcached-sample
  kubectl delete namespace memcached-sample
  ```

在 Amazon EKS 和 Kubernetes 上设置 Java/JMX 示例工作负载

JMX Exporter 是 Prometheus 的官方导出程序，可以将 JMX MBeans 作为 Prometheus 指标进行抓取和公开。有关详细信息，请参阅 prometheus/jmx_exporter。

Container Insights 可以使用 JMX Exporter 从 Java 虚拟机 (JVM)、Java 和 Tomcat (Catalina) 收集预定义的 Prometheus 指标。

默认 Prometheus 抓取配置

默认情况下，支持 Prometheus 的 CloudWatch 代理从 Amazon EKS 或 Kubernetes 集群中的每个 pod 上的 `http://CLUSTER_IP:9404/metrics` 抓取 Java/JMX Prometheus 指标。这是通过 Prometheus `kubernetes_sd_config` 的 `role: pod` 发现来完成的。9404 是 Prometheus 为 JMX Exporter 分配的默认端口。有关 `role: pod` 发现的更多信息，请参阅 pod。您可以将 JMX Exporter 配置为在不同端口或 metrics_path 上公开指标。如果您更改了端口或路径，请更新 CloudWatch 代理 config 映射中的默认 jmx scrape_config。运行以下命令以获取当前 CloudWatch 代理 Prometheus 配置：

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

要更改的字段是 `/metrics` 和 `regex: '.*:9404$'` 字段，如以下示例中突出显示所示。

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
- action: replace
  regex: (.+)
  source_labels:
```

其他 Prometheus 抓取配置

如果您通过 Kubernetes Service 使用 Java/JMX Prometheus 导出器公开在一组 pod 上运行的应用程序，您还可以切换到使用 Prometheus `kubernetes_sd_config` 的 `role: service` 发现或 `role: endpoint` 发现。有关这些发现方法的更多信息，请参阅服务、端点 和 <kubernetes_sd_config>。

这两种服务发现模式提供了更多元标签，这对构建 CloudWatch 指标维度大有裨益。例如，您可以将 `__meta_kubernetes_service_name` 其重新标记为 `Service` 并将其包含在指标维度中。有关自定义 CloudWatch 指标及其维度的更多信息，请参阅 Prometheus 的 CloudWatch 代理配置。

带有 JMX Exporter 的 Docker 镜像

接下来，构建 Docker 镜像。以下各节提供了两个示例 Dockerfiles。

构建镜像后，请将其加载到 Amazon EKS 或 Kubernetes 中，然后运行以下命令以验证 JMX_EXPORTER 是否在端口 9404 上公开 Prometheus 指标。使用正在运行的 pod 名称替换 *$JAR_SAMPLE_TRAFFIC_POD*，并使用您的应用程序命名空间替换 *$JAR_SAMPLE_TRAFFIC_NAMESPACE*。

如果您在具有 Fargate 启动类型的集群上运行 JMX Exporter，则在执行此过程中的步骤之前，您还需要设置 Fargate 配置文件。要设置配置文件，请输入以下命令。将 *MyCluster* 替换为您的集群的名称。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE\
 --name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl
 http://localhost:9404
```

示例：具有 Prometheus 指标的 Apache Tomcat Docker 镜像

默认情况下，Apache Tomcat 服务器公开 JMX MBeans。您可以将 JMX Exporter 与 Tomcat 集成，以便将 JMX MBeans 作为 Prometheus 指标公开。以下示例 Dockerfile 显示了构建测试镜像的步骤：

```
# From Tomcat 9.0 JDK8 OpenJDK
FROM tomcat:9.0-jdk8-openjdk

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter
COPY ./setenv.sh /usr/local/tomcat/bin
COPY your web application.war /usr/local/tomcat/webapps/

RUN chmod  o+x /usr/local/tomcat/bin/setenv.sh
```

```
ENTRYPOINT ["catalina.sh", "run"]
```

下面的列表解释了此 Dockerfile 中的四个 COPY 行。

- 从 https://github.com/prometheus/jmx_exporter 下载最新的 JMX Exporter jar 文件。
- config.yaml 是 JMX Exporter 配置文件。有关更多信息，请参阅 https://github.com/prometheus/jmx_exporter#Configuration。

  以下是 Java 和 Tomcat 的示例配置文件：

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER
```

```
- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `setenv.sh` 是一个 Tomcat 启动脚本，用于启动 JMX Exporter 和 Tomcat，并在本地主机的端口 9404 上公开 Prometheus 指标。它还为 JMX Exporter 提供 `config.yaml` 文件路径。

```
$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/
jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"
```

- 您的 Web 应用程序 .war 是由 Tomcat 加载的 Web 应用程序 `war` 文件。

使用此配置构建 Docker 镜像并将其上载到镜像存储库。

示例：具有 Prometheus 指标的 Java Jar 应用程序 Docker 镜像

以下示例 Dockerfile 显示了构建测试镜像的步骤：

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
```

```
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

下面的列表解释了此 Dockerfile 中的四个 COPY 行。

- 从 https://github.com/prometheus/jmx_exporter 下载最新的 JMX Exporter jar 文件。
- config.yaml 是 JMX Exporter 配置文件。有关更多信息，请参阅 https://github.com/prometheus/jmx_exporter#Configuration。

  以下是 Java 和 Tomcat 的示例配置文件：

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
```

```
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` 是用于启动导出了 Prometheus 指标的 JAR 应用程序的脚本。它还为 JMX Exporter 提供 `config.yaml` 文件路径。

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/
opt/jmx_exporter/config.yaml -cp  /opt/jmx_exporter/SampleJavaApplication-1.0-
SNAPSHOT.jar com.gubupt.sample.app.App
```

- SampleJavaApplication-1.0-SNAPSHOT.jar 是示例 Java 应用程序 jar 文件。将其替换为要监控的 Java 应用程序。

使用此配置构建 Docker 镜像并将其上载到镜像存储库。

使用 Amazon EKS 和 Kubernetes 上的指标导出器设置 HAProxy

HAProxy 是一个开源代理应用程序。有关更多信息，请参阅 HAProxy。

如果您在具有 Fargate 启动类型的集群上运行 HAProxy，则在执行此过程中的步骤之前，您需要设置 Fargate 配置文件。要设置配置文件，请输入以下命令。将 *MyCluster* 替换为您的集群的名称。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

安装带有 Metric Exporter 的 HAProxy 以测试 Container Insights Prometheus 支持

1. 输入以下命令以添加 Helm incubator 存储库：

   ```
   helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
   ```

2. 输入以下命令以创建新的命名空间：

   ```
   kubectl create namespace haproxy-ingress-sample
   ```

3. 输入以下命令来安装 HAProxy：

   ```
   helm install haproxy haproxy-ingress/haproxy-ingress \
   --namespace haproxy-ingress-sample \
   --set defaultBackend.enabled=true \
   --set controller.stats.enabled=true \
   --set controller.metrics.enabled=true \
   --set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
   --set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
   ```

4. 输入以下命令以确认服务的注释：

   ```
   kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
   ```

   您应该看到以下注释。

   ```
   Annotations:   prometheus.io/port: 9101
                  prometheus.io/scrape: true
   ```

卸载 HAProxy

- 输入以下命令：

  ```
  helm uninstall haproxy --namespace haproxy-ingress-sample
  ```

```
kubectl delete namespace haproxy-ingress-sample
```

添加新 Prometheus 抓取目标的教程：Amazon EKS 和 Kubernetes 集群上的 Redis OSS

本教程提供了在 Amazon EKS 和 Kubernetes 上抓取示例 Redis OSS 应用程序的 Prometheus 指标的实践介绍。Redis OSS（https://redis.io/）是一个开源（获得 BSD 许可）的内存数据结构存储，用作数据库、缓存和消息代理。有关更多信息，请参阅 redis。

redis_exporter（获得 MIT 许可证）用于在指定端口（默认值：0.0.0.0:9121）上公开 Redis OSS prometheus 指标。有关更多信息，请参阅 redis_exporter。

本教程使用了以下两个 Docker Hub 存储库中的 Docker 镜像：

- redis
- redis_exporter

安装公开 Prometheus 指标的示例 Redis OSS 工作负载

1. 为示例 Redis OSS 工作负载设置命名空间。

```
REDIS_NAMESPACE=redis-sample
```

2. 如果您在具有 Fargate 启动类型的集群上运行 Redis OSS，则需要设置 Fargate 配置文件。要设置配置文件，请输入以下命令。将 *MyCluster* 替换为您的集群的名称。

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
```

3. 输入以下命令以安装示例 Redis OSS 工作负载。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/redis/redis-traffic-sample.yaml \
| sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \
| kubectl apply -f -
```

4. 安装包括一个名为 `my-redis-metrics` 的服务，该服务在端口 9121 上公开 Redis OSS Prometheus 指标。输入以下命令以获取该服务的详细信息：

```
kubectl describe service/my-redis-metrics  -n $REDIS_NAMESPACE
```

在结果的 Annotations 部分，您将看到与 CloudWatch 代理的 Prometheus 抓取配置相匹配的两个注释，以便其可以自动发现工作负载：

```
prometheus.io/port: 9121
prometheus.io/scrape: true
```

相关的 Prometheus 抓取配置可以在 kubernetes-eks.yaml 或 kubernetes-k8s.yaml 部分的 - job_name: kubernetes-service-endpoints 中找到。

开始在 CloudWatch 中收集 Redis OSS Prometheus 指标

1. 通过输入以下任一命令，下载 kubernetes-eks.yaml 或 kubernetes-k8s.yaml 文件的最新版本。对于具有 EC2 启动类型的 Amazon EKS 集群，请输入此命令。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

对于具有 Fargate 启动类型的 Amazon EKS 集群，请输入此命令。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

对于在 Amazon EC2 实例上运行的 Kubernetes 集群，请输入此命令。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. 使用文本编辑器打开文件，然后找到 cwagentconfig.json 部分。添加以下子部分并保存更改。确保遵循现有模式缩进。

```
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
```

```
  "dimensions": [["Namespace","ClusterName"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

您添加的部分会将 Redis OSS 指标放入 CloudWatch 代理允许列表中。有关这些指标的列表，请参阅以下部分。

3. 如果您已在此集群中部署了具有 Prometheus 支持的 CloudWatch 代理，则必须通过输入以下命令将其删除。

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. 通过输入以下任一命令，使用更新的配置部署 CloudWatch 代理。替换 *MyCluster* 和 *region####*以匹配您的设置。

   对于具有 EC2 启动类型的 Amazon EKS 集群，请输入此命令。

```
kubectl apply -f prometheus-eks.yaml
```

   对于具有 Fargate 启动类型的 Amazon EKS 集群，请输入此命令。

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

对于 Kubernetes 集群，请输入此命令。

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

查看 Redis OSS Prometheus 指标

本教程会将以下指标发送到 CloudWatch 中的 ContainerInsights/Prometheus 命名空间。您可以使用 CloudWatch 控制台查看该命名空间中的指标。

指标名称	尺寸	
redis_net_input_bytes_total	ClusterName、Namespace	
redis_net_output_bytes_total	ClusterName、Namespace	
redis_expired_keys_total	ClusterName、Namespace	
redis_evicted_keys_total	ClusterName、Namespace	
redis_keyspace_hits_total	ClusterName、Namespace	

指标名称	尺寸	
redis_keyspace_misses_total	ClusterName、Namespace	
redis_memory_used_bytes	ClusterName、Namespace	
redis_connected_clients	ClusterName、Namespace	
redis_commands_total	ClusterName、Namespace 、cmd	
redis_db_keys	Cluster、Namespace 、db	

> **Note**
>
> cmd 维度的值可以是 append、client、command、config、dbsize、flushall、get、incr、info、latency 或 slowlog。
> db 维度的值可以从 db0 到 db15。

您还可以为 Redis OSS Prometheus 指标创建 CloudWatch 控制面板。

为 Redis OSS Prometheus 指标创建控制面板

1. 创建环境变量，替换以下值以匹配部署。

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. 输入以下命令以创建控制面板。

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

CloudWatch 代理的 Prometheus 指标类型转换

Prometheus 客户端库提供四种核心指标类型：

- 计数器
- 计量表
- Summary
- 直方图

CloudWatch 代理支持计数器、计量表和汇总指标类型。对直方图指标的支持计划在未来的版本中推出。

CloudWatch 代理会删除具有不受支持的直方图指标类型的 Prometheus 指标。有关更多信息，请参阅录入丢弃的 Prometheus 指标。

计量表指标

Prometheus 计量表指标是表示可以任意上下的单个数值的度量。CloudWatch 代理会抓取计量表指标并直接发送这些值。

计数器指标

Prometheus 计数器指标是一个累积指标，表示单个单调增加的计数器，其值只能增加或重置为零。CloudWatch 代理根据上次抓取计算增量，并将增量值作为日志事件中的指标值发送。因此，CloudWatch 代理将从第二次抓取开始生成一个日志事件，并继续进行后续抓取（如果有）。

汇总指标

Prometheus 汇总指标是一种复杂的指标类型，由多个数据点表示。它提供观察的总数和所有观察值的总和。它在滑动时间窗口内计算可配置的分位数。

汇总指标的总和和计数是累积的，但分位数不是。以下示例显示了分位数的方差。

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

CloudWatch 代理以与处理计数器指标相同的方式处理汇总指标的总和和计数，如上一节所述。CloudWatch 代理会保留最初报告的分位数值。

CloudWatch 代理收集的 Prometheus 指标

具有 Prometheus 支持的 CloudWatch 代理可自动从多个服务和工作负载收集指标。以下部分列出了默认情况下收集的指标。您还可以将代理配置为从这些服务收集更多指标，并从其他应用程序和服务收集 Prometheus 指标。有关可收集的其他指标的更多信息，请参阅 Prometheus 的 CloudWatch 代理配置。

从 Amazon EKS 和 Kubernetes 集群收集的 Prometheus 指标位于 ContainerInsights/Prometheus 命名空间中。从 Amazon ECS 集群收集的 Prometheus 指标位于 ECS/ContainerInsights/Prometheus 命名空间中。

主题

- App Mesh 的 Prometheus 指标
- NGINX 的 Prometheus 指标
- Memcached 的 Prometheus 指标
- Java/JMX 的 Prometheus 指标
- HAProxy 的 Prometheus 指标

App Mesh 的 Prometheus 指标

从 App Mesh 自动收集以下指标。

CloudWatch Container Insights 还可以收集 App Mesh Envoy 访问日志。有关更多信息，请参阅（可选）启用 App Mesh Envoy 访问日志。

Amazon EKS 和 Kubernetes 集群上 App Mesh 的 Prometheus 指标

指标名称	尺寸	
envoy_http_downstream_rq_total	ClusterName、Namespace	
envoy_http_downstream_rq_xx	ClusterName、Namespace ClusterName、Namespace 、envoy_http_conn_manager_prefix、envoy_response_code_class	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName、Namespace	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName、Namespace	
envoy_cluster_membership_healthy	ClusterName、Namespace	
envoy_cluster_membership_total	ClusterName、Namespace	
envoy_server_memory_heap_size	ClusterName、Namespace	
envoy_server_memory_allocated	ClusterName、Namespace	

指标名称	尺寸	
envoy_cluster_upstream_cx_connect_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName、Namespace	
envoy_cluster_upstream_rq_pending_overflow	ClusterName、Namespace	
envoy_cluster_upstream_rq_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName、Namespace	
envoy_cluster_upstream_rq_rx_reset	ClusterName、Namespace	

指标名称	尺寸	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName、Namespace	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName、Namespace	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName、Namespace	

指标名称	尺寸	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName、Namespace	
envoy_cluster_upstream_flow_control_drained_total	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry_success	ClusterName、Namespace	
envoy_cluster_upstream_rq_retry_overflow	ClusterName、Namespace	
envoy_server_live	ClusterName、Namespace	
envoy_server_uptime	ClusterName、Namespace	

Amazon ECS 集群上 App Mesh 的 Prometheus 指标

指标名称	尺寸	
envoy_http_downstream_rq_total	ClusterName、TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_membership_healthy	ClusterName、TaskDefinitionFamily	
envoy_cluster_membership_total	ClusterName、TaskDefinitionFamily	
envoy_server_memory_heap_size	ClusterName、TaskDefinitionFamily	
envoy_server_memory_allocated	ClusterName、TaskDefinitionFamily	
envoy_cluster_upst	ClusterName、TaskDefinitionFamily	

指标名称	尺寸	
ream_cx_connect_timeout		
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_pending_overflow	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_timeout	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_rx_reset	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName、TaskDefinitionFamily	

指标名称	尺寸	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName、TaskDefinitionFamily	

指标名称	尺寸	
envoy_cluster_upstream_flow_control_drained_total	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry_success	ClusterName、TaskDefinitionFamily	
envoy_cluster_upstream_rq_retry_overflow	ClusterName、TaskDefinitionFamily	
envoy_server_live	ClusterName、TaskDefinitionFamily	
envoy_server_uptime	ClusterName、TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName、TaskDefinitionFamily、envoy_http_conn_manager_prefix、envoy_response_code_class ClusterName、TaskDefinitionFamily、envoy_response_code_class	

> **Note**
>
> TaskDefinitionFamily 是网格的 Kubernetes 命名空间。

envoy_http_conn_manager_prefix 的值可以是 ingress、egress 或 admin。envoy_response_code_class 的值可以是 1（代表 1xx）、2（代表 2xx）、3（代表 3xx）、4（代表 4xx）或 5（代表 5xx）。

NGINX 的 Prometheus 指标

从 Amazon EKS 和 Kubernetes 集群上的 NGINX 自动收集以下指标。

指标名称	尺寸	
nginx_ingress_controller_nginx_process_cpu_seconds_total	ClusterName、Namespace 、Service	
nginx_ingress_controller_success	ClusterName、Namespace 、Service	
nginx_ingress_controller_requests	ClusterName、Namespace 、Service	
nginx_ingress_controller_nginx_process_connections	ClusterName、Namespace 、Service	
nginx_ingress_controller_nginx_proce	ClusterName、Namespace 、Service	

指标名称	尺寸	
ss_connections_total		
nginx_ingress_controller_nginx_process_resident_memory_bytes	ClusterName、Namespace 、Service	
nginx_ingress_controller_config_last_reload_successful	ClusterName、Namespace 、Service	
nginx_ingress_controller_requests	ClusterName、Namespace 、Service、status	

Memcached 的 Prometheus 指标

从 Amazon EKS 和 Kubernetes 集群上的 Memcached 中自动收集以下指标。

指标名称	尺寸	
memcached_current_items	ClusterName、Namespace 、Service	
memcached_current_connections	ClusterName、Namespace 、Service	

指标名称	尺寸
memcached_limit_bytes	ClusterName、Namespace 、Service
memcached_current_bytes	ClusterName、Namespace 、Service
memcached_written_bytes_total	ClusterName、Namespace 、Service
memcached_read_bytes_total	ClusterName、Namespace 、Service
memcached_items_evicted_total	ClusterName、Namespace 、Service
memcached_items_reclaimed_total	ClusterName、Namespace 、Service
memcached_commands_total	ClusterName、Namespace 、Service ClusterName、Namespace 、Service、command ClusterName、Namespace 、Service、status、command

Java/JMX 的 Prometheus 指标

在 Amazon EKS 和 Kubernetes 集群上收集的指标

在 Amazon EKS 和 Kubernetes 集群上，Container Insights 可以使用 JMX Exporter 从 Java 虚拟机 (JVM)、Java 和 Tomcat (Catalina) 收集以下预定义的 Prometheus 指标。有关更多信息，请参阅 Github 上的 prometheus/jmx_exporter。

Amazon EKS 和 Kubernetes 集群上的 Java/JMX

指标名称	尺寸	
jvm_classes_loaded	ClusterName ,Namespace	
jvm_threads_current	ClusterName ,Namespace	
jvm_threads_daemon	ClusterName ,Namespace	
java_lang_operatingsystem_totalswapspacesize	ClusterName ,Namespace	
java_lang_operatingsystem_systemcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_processcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_freeswapspacesize	ClusterName ,Namespace	
java_lang_operatingsystem_t	ClusterName ,Namespace	

指标名称	尺寸	
otalphysi calmemorysize		
java_lang _operatin gsystem_f reephysic almemorysize	ClusterName ，Namespace	
java_lang _operatin gsystem_o penfilede scriptorcount	ClusterName ，Namespace	
java_lang _operatin gsystem_a vailablep rocessors	ClusterName ，Namespace	
jvm_memor y_bytes_used	ClusterName 、Namespace 、area	
jvm_memor y_pool_by tes_used	ClusterName 、Namespace 、pool	

> **Note**
>
> area 维度的值可以是 heap 或 nonheap。
> pool 维度的值可以是 Tenured Gen、Compress Class Space、Survivor Space、Eden Space、Code Cache 或 Metaspace。

Amazon EKS 和 Kubernetes 集群上的 Tomcat/JMX

除了上表中的 Java/JMX 指标外，还收集 Tomcat 工作负载的以下指标。

指标名称	尺寸	
catalina_manager_activesessions	ClusterName，Namespace	
catalina_manager_rejectedsessions	ClusterName，Namespace	
catalina_globalrequestprocessor_bytesreceived	ClusterName，Namespace	
catalina_globalrequestprocessor_bytessent	ClusterName，Namespace	
catalina_globalrequestprocessor_requestcount	ClusterName，Namespace	
catalina_globalrequestprocessor_errorcount	ClusterName，Namespace	
catalina_globalreq	ClusterName，Namespace	

指标名称	尺寸	
uestproce ssor_proc essingtime		

Amazon ECS 集群上的 Java/JMX

指标名称	尺寸	
jvm_class es_loaded	ClusterName ，TaskDefinitionFamily	
jvm_threa ds_current	ClusterName ，TaskDefinitionFamily	
jvm_threa ds_daemon	ClusterName ，TaskDefinitionFamily	
java_lang _operatin gsystem_t otalswaps pacesize	ClusterName ，TaskDefinitionFamily	
java_lang _operatin gsystem_s ystemcpuload	ClusterName ，TaskDefinitionFamily	
java_lang _operatin gsystem_p rocesscpuload	ClusterName ，TaskDefinitionFamily	
java_lang _operatin gsystem_f	ClusterName ，TaskDefinitionFamily	

指标名称	尺寸	
reeswapspacesize		
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName ，TaskDefinitionFamily	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName ，TaskDefinitionFamily	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName ，TaskDefinitionFamily	
java_lang_operatingsystem_availableprocessors	ClusterName ，TaskDefinitionFamily	
jvm_memory_bytes_used	ClusterName 、TaskDefinitionFamily、area	
jvm_memory_pool_bytes_used	ClusterName 、TaskDefinitionFamily、pool	

> **Note**
>
> area 维度的值可以是 heap 或 nonheap。
> pool 维度的值可以是 Tenured Gen、Compress Class Space、Survivor Space、Eden Space、Code Cache 或 Metaspace。

Amazon ECS 集群上的 Tomcat/JMX

除上表中的 Java/JMX 指标外，还收集了 Amazon ECS 集群上 Tomcat 工作负载的以下指标。

指标名称	尺寸	
catalina_manager_activesessions	ClusterName，TaskDefinitionFamily	
catalina_manager_rejectedsessions	ClusterName，TaskDefinitionFamily	
catalina_globalrequestprocessor_bytesreceived	ClusterName，TaskDefinitionFamily	
catalina_globalrequestprocessor_bytessent	ClusterName，TaskDefinitionFamily	
catalina_globalrequestprocessor_requestcount	ClusterName，TaskDefinitionFamily	

指标名称	尺寸	
catalina_globalrequestprocessor_errorcount	ClusterName ，TaskDefinitionFamily	
catalina_globalrequestprocessor_processingtime	ClusterName ，TaskDefinitionFamily	

HAProxy 的 Prometheus 指标

从 Amazon EKS 和 Kubernetes 集群上的 HAProxy 自动收集以下指标。

收集的指标取决于您使用的 HAProxy Ingress 版本。有关 HAProxy Ingress 及其版本的更多信息，请参阅 haproxy-ingress。

指标名称	尺寸	可用性
haproxy_backend_bytes_in_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_backend_bytes_out_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_backend_connection_errors_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_backend_co	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress

指标名称	尺寸	可用性
nnections_total		
haproxy_backend_current_sessions	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_backend_http_responses_total	ClusterName 、Namespace 、Service、code、backend	所有版本的 HAProxy Ingress
haproxy_backend_status	ClusterName 、Namespace 、Service	仅在 HAProxy Ingress 0.10 或更高版本中
haproxy_backend_up	ClusterName 、Namespace 、Service	仅在 HAProxy Ingress 0.10 的更早版本中
haproxy_frontend_bytes_in_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_frontend_bytes_out_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_frontend_connections_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_frontend_current_sessions	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress

指标名称	尺寸	可用性
haproxy_frontend_http_requests_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_frontend_http_responses_total	ClusterName 、Namespace 、Service、code、frontend	所有版本的 HAProxy Ingress
haproxy_frontend_request_errors_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress
haproxy_frontend_requests_denied_total	ClusterName 、Namespace 、Service	所有版本的 HAProxy Ingress

Note

code 维度的值可以是 1xx、2xx、3xx、4xx、5xx 或 other。
backend 维度的值可以是：

- 适用于 HAProxy Ingress 0.0.27 或更早版本的 http-default-backend、http-shared-backend 或 httpsback-shared-backend。
- 适用于高于 HAProxy Ingress 0.0.27 版本的 _default_backend。

frontend 维度的值可以是：

- 适用于 HAProxy Ingress 0.0.27 或更早版本的 httpfront-default-backend、httpfront-shared-frontend 或 httpfronts。
- 适用于高于 HAProxy Ingress 0.0.27 版本的 _front_http 或 _front_https。

查看 Prometheus 指标

您可以监控所有 Prometheus 指标并发出告警，包括来自 App Mesh、NGINX、Java/JMX、Memcached 和 HAProxy 的精选预聚合指标，以及任何其他您可能已添加的手动配置的 Prometheus 导出程序。有关从其他 Prometheus 导出程序收集指标的更多信息，请参阅添加新 Prometheus 抓取目标的教程：Prometheus API 服务器指标。

在 CloudWatch 控制台中，Container Insights 提供以下预构建报告：

- 对于 Amazon EKS 和 Kubernetes 集群，有针对 App Mesh、NGINX、HAPROXY、Memcached 和 Java/JMX 的预构建报告。
- 对于 Amazon ECS 集群，有针对 App Mesh 和 Java/JMX 的预构建报告。

Container Insights 还为 Container Insights 从中收集策管指标的每个工作负载提供自定义控制面板。您可以从 GitHub 下载这些控制面板

查看所有 Prometheus 指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 在命名空间列表中，选择 ContainerInsights/Prometheus 或 ECS/ContainerInsights/Prometheus。
4. 在以下列表中选择一组维度。然后选中您要查看的指标旁边的复选框。

查看有关 Prometheus 指标的预构建报告

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择性能监控。
3. 在页面顶部附近的下拉框中，选择任一 Prometheus 选项。

   在另一个下拉框中，选择要查看的集群

我们还为 NGINX、App Mesh、Memcached、HAProxy 和 Java/JMX 提供了自定义控制面板。

使用 Amazon 提供的自定义控制面板

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 Dashboards（控制面板）。
3. 选择创建控制面板。输入新控制面板的名称，然后选择创建控制面板。
4. 在添加到该控制面板中，选择取消。
5. 依次选择操作、编辑控制面板。
6. 下载以下 JSON 文件之一：
   - Github 上的 NGINX 自定义控制面板源代码。
   - Github 上的 App Mesh 自定义控制面板源代码。
   - Github 上的 Memcached 自定义控制面板源代码
   - Github 上的 HAProxy-Ingress 自定义控制面板源代码
   - Github 上的 Java/JMX 自定义控制面板源代码。
7. 使用文本编辑器打开下载的 JSON 文件，然后执行以下更改：
   - 将所有 `{{YOUR_CLUSTER_NAME}}` 字符串替换为您的集群的确切名称。请勿在文本之前或之后添加空格。
   - 将所有 `{{YOUR_REGION}}` 字符串替换为正在运行集群的 AWS 区域。例如，**`us-west-1`** 请勿在文本之前或之后添加空格。
   - 将所有 `{{YOUR_NAMESPACE}}` 字符串替换为您的工作负载的确切命名空间。
   - 将所有 `{{YOUR_SERVICE_NAME}}` 字符串替换为您的工作负载的确切服务名称。例如，**`haproxy-haproxy-ingress-controller-metrics`**
8. 复制整个 JSON blob 并将其粘贴到 CloudWatch 控制台的文本框中，替换框中已有的内容。
9. 选择更新、保存控制面板。

Prometheus 指标故障排除

本节提供有关排除 Prometheus 指标设置故障排除的帮助。

主题

- 对 Amazon ECS 上的 Prometheus 指标进行故障排除
- Amazon EKS 和 Kubernetes 集群上的 Prometheus 指标故障排除

对 Amazon ECS 上的 Prometheus 指标进行故障排除

本节提供有关对 Amazon ECS 集群上的 Prometheus 指标设置进行故障排除的帮助。

我没有看到发送到 CloudWatch Logs 的 Prometheus 指标

Prometheus 指标在日志组 /aws/ecs/containerinsights/cluster-name/Prometheus 中摄取作为日志事件。如果未创建日志组或未将 Prometheus 指标发送到日志组，您需要首先检查 CloudWatch 代理是否已成功发现 Prometheus 目标。然后检查 CloudWatch 代理的安全组和权限设置。以下步骤会指导您进行调试。

步骤 1：启用 CloudWatch 代理调试模式

首先，通过将以下粗体行添加到 AWS CloudFormation 模版文件、`cwagent-ecs-prometheus-metric-for-bridge-host.yaml` 或 `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` 中并保存文件，将 CloudWatch 代理更改为调试模式。然后保存文件。

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

针对现有堆栈创建一个新的 AWS CloudFormation 变更集。将变更集中的其他参数设置为与现有 AWS CloudFormation 堆栈中相同的值。以下示例适用于使用 EC2 启动类型和桥式网络模式安装在 Amazon ECS 集群中的 CloudWatch 代理。

```
ECS_NETWORK_MODE=bridge
 CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
 \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
```

```
    --change-set-name $NEW_CHANGESET_NAME
```

转至 AWS CloudFormation 控制台查看新的变更集，请参阅 $NEW_CHANGESET_NAME。应该对 CWAgentConfigSSMParameter 资源应用一项更改。通过输入以下命令执行变更集并重新启动 CloudWatch 代理任务。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

等待大约 10 秒，然后输入以下命令。

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

步骤 2：检查 ECS 服务发现日志

CloudWatch 代理的 ECS 任务定义默认启用以下部分中的日志。日志将发送到 /ecs/ecs-cwagent-prometheus 日志组中的 CloudWatch Logs。

```
LogConfiguration:
  LogDriver: awslogs
    Options:
      awslogs-create-group: 'True'
      awslogs-group: "/ecs/ecs-cwagent-prometheus"
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

根据字符串 ECS_SD_Stats 筛选日志以获取与 ECS 服务发现相关的指标，如下例所示。

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2
```

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

特定 ECS 服务发现周期的每个指标的含义如下：

- AWSCLI_DescribeContainerInstances – `ECS::DescribeContainerInstances` 进行的 API 调用次数。
- AWSCLI_DescribeInstancesRequest – `ECS::DescribeInstancesRequest` 进行的 API 调用次数。
- AWSCLI_DescribeTaskDefinition – `ECS::DescribeTaskDefinition` 进行的 API 调用次数。
- AWSCLI_DescribeTasks – `ECS::DescribeTasks` 进行的 API 调用次数。
- AWSCLI_ListTasks – `ECS::ListTasks` 进行的 API 调用次数。
- ExporterDiscoveredTargetCount – 已发现并成功导出到容器内的目标结果文件中的 Prometheus 目标数量。
- LRUCache_Get_EC2MetaData – 从缓存中检索容器实例元数据的次数。
- LRUCache_Get_TaskDefinition – 从缓存中检索 ECS 任务定义元数据的次数。
- LRUCache_Size_ContainerInstance – 缓存在内存中的唯一容器实例元数据的数量。
- LRUCache_Size_TaskDefinition – 缓存在内存中的唯一 ECS 任务定义的数量。
- 延迟 – 服务发现周期所需的时间。

检查 `ExporterDiscoveredTargetCount` 的值以查看发现的 Prometheus 目标是否符合您的预期。如果不符合，可能的原因如下：

- ECS 服务发现的配置可能与应用程序的设置不匹配。对于基于 docker 标签的服务发现，您的目标容器可能并未在 CloudWatch 代理中配置必要的 docker 标签以将其自动发现。对于 ECS 任务定义 ARN 基于正则表达式的服务发现，CloudWatch 代理中的正则表达式设置可能与应用程序的任务定义不匹配。
- CloudWatch 代理的 ECS 任务角色可能没有权限检索 ECS 任务的元数据。检查 CloudWatch 代理是否已被授予以下只读权限：
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeTasks`

- `ecs:DescribeTaskDefinition`

步骤 3：检查网络连接以及 ECS 任务角色策略

如果即使 `Exporter_DiscoveredTargetCount` 的值表明已发现 Prometheus 目标，仍然没有日志事件发送到目标 CloudWatch Logs 日志组，这可能是由以下任一原因引起的：

- CloudWatch 代理可能无法连接到 Prometheus 目标端口。检查 CloudWatch 代理背后的安全组设置。私有 IP 应允许 CloudWatch 代理连接到 Prometheus 导出器端口。
- CloudWatch 代理的 ECS 任务角色可能没有 CloudWatchAgentServerPolicy 托管策略。CloudWatch 代理的 ECS 任务角色需要具有此策略才能将 Prometheus 指标作为日志事件发送。如果您使用示例 AWS CloudFormation 模板自动创建 IAM 角色，则 ECS 任务角色和 ECS 执行角色都会被授予执行 Prometheus 监控的最低权限。

Amazon EKS 和 Kubernetes 集群上的 Prometheus 指标故障排除

本节提供有关对 Amazon EKS 和 Kubernetes 集群上的 Prometheus 指标设置进行故障排除的帮助。

Amazon EKS 上的一般故障排除步骤

要确认 CloudWatch 代理正在运行，请输入以下命令。

```
kubectl get pod -n amazon-cloudwatch
```

输出应包含其 `NAME` 列中包含 `cwagent-prometheus-`*`id`* 并且 `STATUS column.`中包含 `Running` 的行。

要显示有关正在运行的 pod 的详细信息，请输入以下命令。将 *`pod-name`* 替换为名称以 `cw-agent-prometheus` 开头的 pod 的完整名称。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

如果您安装了 CloudWatch Container Insights，则可以使用 CloudWatch Logs Insights 从收集 Prometheus 指标的 CloudWatch 代理查询日志。

查询应用程序日志

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。

2. 在导航窗格中，选择 CloudWatch Logs Insights。
3. 选择应用程序日志的日志组 /aws/containerinsights/***cluster-name***/application
4. 使用以下查询替换搜索查询表达式，然后选择运行查询

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,
 kubernetes.pod_name, log |
filter haskubernetes_pod_name and kubernetes.pod_name like /cwagent-prometheus
```

您还可以确认 Prometheus 指标和元数据是否摄取作为 CloudWatch Logs 事件。

确认正在摄取 Prometheus 数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 CloudWatch Logs Insights。
3. 选择 /aws/containerinsights/***cluster-name***/prometheus
4. 使用以下查询替换搜索查询表达式，然后选择运行查询

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

录入丢弃的 Prometheus 指标

此版本不会收集直方图的 Prometheus 指标。您可以使用 CloudWatch 代理来检查是否因为任何 Prometheus 指标是直方图指标而丢弃了这些指标。您还可以记录由于是直方图指标，因此丢弃而未发送到 CloudWatch 的前 500 个 Prometheus 指标列表。

要查看是否丢弃了任何指标，请输入以下命令：

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

如果丢弃了任何指标，您将在 /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log 文件中看到以下行。

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are
 supported.
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

如果您看到这些行并想知道丢弃了哪些指标，请使用以下步骤。

记录已丢弃的 Prometheus 指标列表

1. 通过将以下粗体行添加到 prometheus-eks.yaml 或 prometheus-k8s.yaml 文件中并保存文件，将 CloudWatch 代理更改为调试模式。

```
{
      "agent": {
        "debug": true
      },
```

该文件的这一部分随后应该如下所示：

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

2. 通过输入以下命令重新安装 CloudWatch 代理以启用调试模式：

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
kubectl apply -f prometheus.yaml
```

丢弃的指标将在 CloudWatch 代理 pod 中录入。

3. 要从 CloudWatch 代理 pod 检索日志，请输入以下命令：

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

或者，如果您安装了 Container Insights Fluentd 日志记录，日志也会保存在 CloudWatch Logs 日志组 /aws/containerinsights/***cluster_name***/application 中。

要查询这些日志，您可以按照Amazon EKS 上的一般故障排除步骤中的步骤查询应用程序日志。

在哪里摄取 Prometheus 指标作为 CloudWatch Logs 日志事件？

CloudWatch 代理为每个 Prometheus 抓取任务配置创建一个日志流。例如，在 prometheus-eks.yaml 和 prometheus-k8s.yaml 文件中，job_name: 'kubernetes-pod-appmesh-

envoy' 行将抓取 App Mesh 指标。Prometheus 目标被定义为 kubernetes-pod-appmesh-envoy。因此，所有 App Mesh Prometheus 指标都作为 CloudWatch Logs 事件摄取到名为 /aws/containerinsights/cluster-name/Prometheus 的日志组下的 kubernetes-pod-appmesh-envoy 日志流中。

我在 CloudWatch 指标中没有看到 Amazon EKS 或 Kubernetes Prometheus 指标

首先，确保 Prometheus 指标在日志组 /aws/containerinsights/cluster-name/Prometheus 中摄取作为日志事件。使用在哪里摄取 Prometheus 指标作为 CloudWatch Logs 日志事件？中的信息可帮助您检查目标日志流。如果未创建日志流或日志流中没有新的日志事件，请检查以下内容：

- 检查 Prometheus 指标导出程序端点是否正确设置
- 检查 CloudWatch 代理 YAML 文件 config map: cwagent-prometheus 部分中的 Prometheus 抓取配置是否正确。该配置应与 Prometheus 配置文件中的配置相同。有关更多信息，请参阅 Prometheus 文档中的 <scrape_config>。

如果正确摄取了 Prometheus 指标作为日志事件，请检查嵌入式指标格式设置是否已添加到日志事件以生成 CloudWatch 指标。

```
"CloudWatchMetrics":[
   {
      "Metrics":[
         {
            "Name":"envoy_http_downstream_cx_destroy_remote_active_rq"
         }
      ],
      "Dimensions":[
         [
            "ClusterName",
            "Namespace"
         ]
      ],
      "Namespace":"ContainerInsights/Prometheus"
   }
],
```

有关嵌入式指标格式的更多信息，请参阅规范：嵌入式指标格式 。

如果日志事件中没有嵌入式指标格式，请检查在 CloudWatch 代理安装 YAML 文件的 `config map: prometheus-cwagentconfig` 部分是否正确配置了 `metric_declaration` 部分。有关更多信息，请参阅 添加新 Prometheus 抓取目标的教程：Prometheus API 服务器指标。

与 Application Insights 集成

Amazon CloudWatch Application Insights 可帮助您监控应用程序，并在应用程序资源和技术堆栈中指定和设置关键指标、日志和告警。有关更多信息，请参阅 Amazon CloudWatch Application Insights。

您可以启用 Application Insights，从容器化应用程序和微服务中收集其他数据。如果您尚未执行此操作，可以通过在 Container Insights 控制面板中的性能视图下方选择 Auto-configure Application Insights（自动配置 Application Insights）将其启用。

如果您已设置 CloudWatch Application Insights 来监控容器化应用程序，Application Insights 控制面板将显示在 Container Insights 控制面板下方。

有关 Application Insights 和容器化应用程序的更多信息，请参阅 为 Amazon ECS 和 Amazon EKS 资源监控启用 Application Insights。

在 Container Insights 中查看 Amazon ECS 生命周期事件

您可以在 Container Insights 控制台中查看 Amazon ECS 生命周期事件。这样有助于您在一个视图中将容器指标、日志和事件关联在一起，从而便于您更完整地了解运行情况。

事件包括容器实例状态更改事件、任务状态更改事件和服务操作事件。它们会由 Amazon ECS 自动发送到 Amazon EventBridge，同时以事件日志的形式收集到 CloudWatch 中。有关这些事件的更多信息，请参阅 Amazon ECS 事件。

Amazon ECS 生命周期事件适用标准 Continer Insights 定价。有关更多信息，请参阅 Amazon CloudWatch 定价。

要配置生命周期事件表并为集群创建规则，您必须拥有 `events:PutRule`、`events:PutTargets` 和 `logs:CreateLogGroup` 权限。您还必须确保有资源策略让 EventBridge 能够创建日志流并将日志发送到 CloudWatch Logs。如果此资源策略不存在，则可以输入以下命令进行创建：

```
aws --region region logs put-resource-policy --policy-name 'EventBridgeCloudWatchLogs'
 --policy-document '{
    "Statement": [
        {
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
```

```
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
            },
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/events/ecs/
containerinsights/*:*",
            "Sid": "TrustEventBridgeToStoreECSLifecycleLogEvents"
        }
    ],
    "Version": "2012-10-17"
}'
```

您可以使用以下命令来检查您是否已有此策略，并确认附加策略的操作已正确执行。

```
aws logs describe-resource-policies --region region --output json
```

要查看生命周期事件表，您必须拥有 `events:DescribeRule`、`events:ListTargetsByRule` 和 `logs:DescribeLogGroups` 权限。

要在 CloudWatch Container Insights 控制台中查看 Amazon ECS 生命周期事件

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 选择 Insights、Container Insights。
3. 选择查看性能控制面板。
4. 在下一个下拉列表中，选择 ECS Clusters（ECS 集群）、ECS Services（ECS 服务）或 ECS Tasks（ECS 任务）。
5. 如果您在上一步中选择了 ECS Services（ECS 服务）或 ECS Tasks（ECS 任务），则选择 Lifecycle events（生命周期事件）选项卡。
6. 在页面底部，如果您看到 Configure lifecycle events（配置生命周期事件），选择它来为您的集群创建 EventBridge 规则。

   这些事件会显示在 Container Insights 窗格的下方和 Application Insights 部分的上方。要对这些事件进行额外分析并创建其他可视化效果，请在生命周期事件表中选择 View in Logs Insights（在 Logs Insights 中查看）。

Container Insights 问题排查

如果在使用 Container Insights 时遇到问题，以下部分可为您提供帮助。

在 Amazon EKS 或 Kubernetes 上部署失败

如果未在 Kubernetes 集群上正确部署该代理，请尝试执行以下操作：

- 运行以下命令以获取 pod 列表。

```
kubectl get pods -n amazon-cloudwatch
```

- 运行以下命令并在输出底部检查事件。

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- 运行以下命令以检查日志。

```
kubectl logs pod-name -n amazon-cloudwatch
```

未经授权的 panic：无法从 kubelet 检索 cadvisor 数据

如果您的部署失败，并显示错误 `Unauthorized panic: Cannot retrieve cadvisor data from kubelet`，则您的 kubelet 可能未启用 Webhook 授权模式。Container Insights 需要此模式。有关更多信息，请参阅 验证 先决条件。

在 Amazon ECS 上已删除和重新创建的集群上部署 Container Insights

如果删除未启用 Container Insights 的现有 Amazon ECS 集群，并使用相同名称重新创建它，则无法在重新创建此集群时在此新集群上启用 Container Insights。您可以通过重新创建来启用它，然后输入以下命令：

```
aws ecs update-cluster-settings --cluster myCICluster --settings
 name=containerInsights,value=enabled
```

端点无效错误

如果您看到类似于以下内容的错误消息，请检查以确保已将正在使用的命令中的所有占位符（例如 *cluster-name* 和 *region-name*）替换为正确的部署信息。

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:
 invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://
```

```
logs.{{region_name}}.amazonaws.com/\", Err:\"{\"}, &awserr.baseError{code:
\"InvalidEndpointURL\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)
(0xc0008723c0)}}\n",
```

指标未显示在控制台中

如果您在 AWS Management Console中未看到任何 Container Insights 指标，请确保已完成 Container Insights 的设置。在完全设置 Container Insights 之前，不会显示指标。有关更多信息，请参阅 设置 Container Insights。

升级集群后，Amazon EKS 或 Kubernetes 上缺少 Pod 指标

如果在您将 CloudWatch 代理作为进程守护程序集部署在新的或升级的集群上后，全部或部分容器组（pod）指标丢失，或者您看到包含 `W! No pod metric collected` 消息的错误日志，则本节可能很有帮助。

这些错误可能是由容器运行时的更改引起的，例如 containerd 或 docker systemd cgroup 驱动程序。您通常可以通过更新部署清单来解决此问题，以便将主机的 containerd 套接字挂载到容器中。请参见以下示例：

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-
insights/blob/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/cwagent/cwagent-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  template:
    spec:
      containers:
        - name: cloudwatch-agent
# ...
          # Don't change the mountPath
          volumeMounts:
# ...
            - name: dockersock
              mountPath: /var/run/docker.sock
              readOnly: true
            - name: varlibdocker
              mountPath: /var/lib/docker
```

```
              readOnly: true
            - name: containerdsock # NEW mount
              mountPath: /run/containerd/containerd.sock
              readOnly: true
# ...
      volumes:
# ...
        - name: dockersock
          hostPath:
            path: /var/run/docker.sock
        - name: varlibdocker
          hostPath:
            path: /var/lib/docker
        - name: containerdsock # NEW volume
          hostPath:
            path: /run/containerd/containerd.sock
```

使用 Bottlerocket for Amazon EKS 时没有 pod 指标

Bottlerocket 是一个基于 Linux 的开源操作系统，AWS 专为运行容器而构建。

Bottlerocket 在主机上使用不同的 `containerd` 路径，因此您需要将卷更改为其位置。如果未这样设置，包含 `W! No pod metric collected` 的日志会显示错误。请参阅以下示例。

```
volumes:
  # ...
    - name: containerdsock
      hostPath:
        # path: /run/containerd/containerd.sock
        # bottlerocket does not mount containerd sock at normal place
        # https://github.com/bottlerocket-os/bottlerocket/
commit/91810c85b83ff4c3660b496e243ef8b55df0973b
        path: /run/dockershim.sock
```

将 containerd 运行时用于 Amazon EKS 或 Kubernetes 时，没有容器文件系统指标

这是一个已知问题，正在由社群贡献者解决。有关更多信息，请参阅 GitHub 上的 containerd 的磁盘使用指标和 containerd 的 cadvisor 不支持容器文件系统指标。

收集 Prometheus 指标时，CloudWatch 代理的日志卷意外增加

这是 CloudWatch 代理的 1.247347.6b250880 版本中推出的回归。此回归已在代理的更新版本中得到修复。它的影响仅限于客户收集 CloudWatch 代理本身的日志并且仍在使用 Prometheus 的情况。有关更多信息，请参阅 GitHub 上的 [prometheus] 代理正在日志中打印所有收集的指标。

未从 Dockerhub 找到发布说明中提到的最新 docker 镜像

在内部开始实际发布之前，我们会更新 Github 上的发布说明和标签。在 Github 上增加版本号后，通常需要 1 – 2 周才能在注册表上看到最新的 Docker 镜像。CloudWatch 代理容器镜像不会在夜间发布。您可以在以下位置直接从源代码构建镜像：https://github.com/aws/amazon-cloudwatch-agent/tree/main/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile

CloudWatch 代理上的 CrashLoopBackoff 错误

如果您看到 CloudWatch 代理出现 `CrashLoopBackOff` 错误，请确保您的 IAM 权限设置正确。有关更多信息，请参阅 验证 先决条件。

CloudWatch 代理或 Fluentd 容器组（pod）卡在待处理状态

如果您有一个 CloudWatch 代理或 Fluentd 容器组（pod）卡在 `Pending` 状态或出现 `FailedScheduling` 错误，请根据代理所需的内核数量和 RAM 量确定您的节点是否有足够的计算资源。使用以下命令描述此 pod：

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

构建您自己的 CloudWatch 代理 Docker 镜像

您可以通过引用位于以下网址的 Dockerfile 来构建自己的 CloudWatch 代理 Docker 镜像：https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/latest/cloudwatch-agent-dockerfile/Dockerfile。

Dockerfile 支持直接使用 `docker buildx` 构建多架构镜像。

在容器中部署其他 CloudWatch 代理功能

您可以使用 CloudWatch 代理在容器中部署其他监控功能。这些特征如下所示：

- 嵌入式指标格式 – 有关更多信息，请参阅 在日志中嵌入指标。

- StatsD – 有关更多信息，请参阅 使用 StatsD 检索自定义指标 。

说明和必要的文件位于 GitHub 上的以下位置：

- 对于 Amazon ECS 容器，请参阅基于部署模式的 Amazon ECS 任务定义示例。
- 对于 Amazon EKS 和 Kubernetes 容器，请参阅基于部署模式的 Kubernetes YAML 文件示例。

Lambda Insights

CloudWatch Lambda Insights 是针对在 AWS Lambda 上运行的无服务器应用程序的监控和故障排除解决方案。此解决方案收集、聚合和汇总系统级指标，包括 CPU 时间、内存、磁盘和网络。它还收集、聚合和汇总诊断信息（如冷启动和 Lambda 工件关闭），以帮助您隔离 Lambda 函数的问题并快速解决这些问题。

Lambda Insights 使用作为 Lambda 层提供的全新 CloudWatch Lambda 扩展程序。在 Lambda 函数上安装此扩展程序时，其会收集系统级指标，并为每次调用该 Lambda 函数发出一个性能日志事件。CloudWatch 使用嵌入式指标格式从日志事件中提取指标。

有关 Lambda 扩展程序的更多信息，请参阅使用 AWS Lambda 扩展程序。有关嵌入式指标格式的更多信息，请参阅在日志中嵌入指标。

您可以将 Lambda Insights 与使用 Lambda 运行时（支持 Lambda 扩展程序）的 Lambda 函数一起使用。有关这些运行时的列表，请参阅 Lambda 扩展程序 API。

定价

对于为 Lambda Insights 启用的每个 Lambda 函数，您只需为用于指标和日志的流量付费。有关定价示例，请参阅 Amazon CloudWatch 定价。

您需要按 Lambda 扩展程序所占用的执行时间（以 1 毫秒为增量计费）付费。有关 Lambda 定价的更多信息，请参阅 AWS Lambda 定价。

开始使用 Lambda Insights

要在 Lambda 函数上启用 Lambda Insights，您可以使用 Lambda 控制台中的一键切换。除此之外，您也可以使用 AWS CLI、AWS CloudFormation、AWS Serverless Application Model CLI 或 AWS Cloud Development Kit (AWS CDK)。

下面几节提供了完成这些步骤的详细说明。

主题

- Lambda Insights 扩展程序的可用版本
- 使用控制台对现有 Lambda 函数启用 Lambda Insights
- 使用 AWS CLI 对现有 Lambda 函数启用 Lambda Insights
- 使用 AWS SAM CLI 对现有 Lambda 函数启用 Lambda Insights
- 使用 AWS CloudFormation 对现有 Lambda 函数启用 Lambda Insights
- 使用 AWS CDK 对现有 Lambda 函数启用 Lambda Insights
- 使用无服务器框架对现有 Lambda 函数启用 Lambda Insights
- 对 Lambda 容器映像部署启用 Lambda Insights
- 更新函数的 Lambda Insights 扩展程序版本

Lambda Insights 扩展程序的可用版本

本节列出了 Lambda Insights 扩展程序的版本，以及要在每个 AWS 区域用于这些扩展程序的 ARN。

主题

- x86-64 平台
- ARM64 平台

x86-64 平台

本节列出了适用于 x84-64 平台的 Lambda Insights 扩展程序版本，以及要在每个 AWS 区域用于这些扩展程序的 ARN。

> **Important**
>
> Lambda Insights 扩展版 1.0.317.0 和更高版本不支持 Amazon Linux 1。

1.0.333.0

版本 1.0.333.0 包括错误修复。

适用于版本 1.0.333.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:53
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:53
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:53
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:53
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:44
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:44
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:26
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:30
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:21
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:51
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:34
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:52

区域	ARN
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:53
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:53
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:80
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:52
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:13
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:43
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:43
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:53
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:53
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:53
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:44
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:52
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:28

区域	ARN
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:50
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:27
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:21
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:44
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:27
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:52
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:21
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:21

1.0.317.0

版本 1.0.317.0 包括取消对 Amazon Linux 1 平台的支持和错误修复。它还包括对 AWS GovCloud (US) 区域的支持。

适用于版本 1.0.317.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:52

区域	ARN
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:52
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:52
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:52
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:43
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:43
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:25
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:29
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:20
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:50
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:33
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:51
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:52
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:52

区域	ARN
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:79
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:51
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:12
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:42
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:42
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:52
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:52
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:52
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:43
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:51
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:27
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:49
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:26

区域	ARN
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:20
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:43
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:26
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:51
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:19
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:19

1.0.295.0

版本 1.0.295.0 包含对所有兼容的运行时的依赖项更新。

适用于版本 1.0.295.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:51
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:51
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:51

区域	ARN
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:51
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:42
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:42
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:24
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:28
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:19
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:49
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:32
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:50
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:51
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:51
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:78
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:50

区域	ARN
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:11
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:41
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:41
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:51
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:51
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:51
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:42
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:50
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:26
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:48
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:25
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:19
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:42

区域	ARN
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:25
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:50

1.0.275.0

版本 1.0.275.0 包含对所有兼容的运行时系统的重要依赖项更新。

适用于版本 1.0.275.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:49
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:49
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:49
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:49
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:40
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:40
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:22

区域	ARN
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:26
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:17
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:47
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:30
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:48
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:49
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:49
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:76
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:48
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:9
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:39
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:39
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:49

区域	ARN
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:49
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:49
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:40
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:48
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:24
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:46
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:23
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:17
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:40
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:23
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:48

1.0.273.0

版本 1.0.273.0 包含了对所有兼容运行时系统的重要错误修复，并增加了对加拿大西部（卡尔加里）区域的支持。

适用于版本 1.0.273.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:45
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:45
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:45
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:45
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:35
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:35
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:17
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:21
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:12
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:43
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:26

区域	ARN
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:44
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:45
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:45
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:72
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:44
加拿大西部（卡尔加里）	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:4
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:36
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:36
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:45
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:45
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:45
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:35
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:44

区域	ARN
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:19
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:42
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:17
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:12
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:35
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:18
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:44

1.0.229.0

版本 1.0.229.0 包含了对所有兼容运行时系统的重要错误修复，并增加了对以色列（特拉维夫）区域的支持。

适用于版本 1.0.229.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:38
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:38

区域	ARN
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:38
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:38
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:28
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:28
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:10
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:14
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:5
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:36
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:19
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:37
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:38
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:38
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:60

区域	ARN
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:37
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:29
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:29
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:38
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:38
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:38
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:28
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:37
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:12
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:35
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:11
以色列（特拉维夫）	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:5
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:28

区域	ARN
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:11
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:37

1.0.178.0

版本 1.0.178.0 增加了对以下 AWS 区域的支持。

- 亚太地区（海得拉巴）
- 亚太地区（雅加达）
- 欧洲（西班牙）
- 欧洲（苏黎世）
- 中东（阿联酋）

适用于版本 1.0.178.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:35
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:33
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:33
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:33

区域	ARN
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:25
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:25
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:8
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:11
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:31
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:32
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:33
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:33
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:50
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:32
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:26
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:26

区域	ARN
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:35
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:33
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:33
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:25
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:32
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:10
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:30
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:7
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:25
中东（阿联酋）	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:9
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:32

1.0.143.0

版本 1.0.143.0 包括与 Python 3.7 和 Go 1.x 兼容性方面的错误修复。Python 3.6 Lambda 运行时已被弃用。有关更多信息，请参阅 Lambda 运行时。

适用于版本 1.0.143.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:21
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:21
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:20
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:21
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:13
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:13
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:21
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:20
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:21
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:21

区域	ARN
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:32
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:20
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:14
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:14
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:21
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:21
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:21
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:13
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:20
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:20
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:13
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:20

1.0.135.0

版本 1.0.135.0 包含针对 Lambda Insights 如何收集以及报告磁盘和文件描述符使用情况的错误修复。在此扩展程序的早期版本中，tmp_free 指标报告了函数运行时 /tmp 目录中的最大可用空间。此版本将指标更改为报告最小值，从而使其在评估磁盘使用情况时更有实用意义。有关 tmp 目录存储配额的更多信息，请参阅 Lambda 配额。

版本 1.0.135.0 现在还报告文件描述符使用情况（fd_use 和 fd_max）作为进程范围内的最大值，而不是报告操作系统级别。

适用于版本 1.0.135.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:18
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:18
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:18
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:18
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:11
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:11
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:18
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:1

区域	ARN
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:18
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:18
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:18
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:25
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:18
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:11
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:11
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:18
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:18
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:18
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:11
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:18
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:18

区域	ARN
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:11
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:18

1.0.119.0

适用于 1.0.119.0 版本的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:16
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:16
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:16
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:16
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:9
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:9
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:16
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:16

区域	ARN
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:16
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:16
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:23
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:16
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:9
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:9
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:16
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:16
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:16
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:9
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:16
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:16
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:9

区域	ARN
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:16

1.0.98.0

此版本删除了不必要的日志记录功能，并解决了 AWS Serverless Application Model CLI 本地调用的一个问题。有关此问题的更多信息，请参阅在具有“sam 本地调用”的超时中添加 LambdaInsightsExtension 结果。

适用于版本 1.0.98.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:14
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:14
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:14
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:8
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:8
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:14

区域	ARN
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:14
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:14
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:14
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:14
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:14
中国（北京）	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:8
中国（宁夏）；	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:8
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:14
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:14
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:14
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:8
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:14
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:14

区域	ARN
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:8
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:14

1.0.89.0

此版本将性能事件时间戳更正为始终表示函数调用的开始。

适用于版本 1.0.89.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:12
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:12
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:12
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:12
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:12
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:12
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:12

区域	ARN
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:12
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:12
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:12
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:12
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:12
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:12
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:12
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:12
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:12

1.0.86.0

此扩展程序的 1.0.54.0 版本有时会误报内存指标，有时这些指标会高于 100%。1.0.86.0 版本使用与 Lambda 平台指标相同的事件数据，纠正了内存度量问题。也就是说，您可能会发现所记录的内存指标值发生了巨大变化。这是通过使用新的 Lambda Logs API 来实现的。它可以更准确地度量 Lambda 沙盒内存使用情况。但需要注意的是，如果函数沙盒超时并随后崩溃，Lambda Logs API 将无法提供平台报告事件。在本例中，Lambda Insights 无法记录调用指标。有关 Lambda Logs API 的更多信息，请参阅 AWS Lambda Logs API。

版本 1.0.86.0 中的新功能

- 使用 Lambda Logs API 更正内存指标。此功能解决了上一个内存统计数据大于 100% 的问题。
- 引入新的 CloudWatch 指标：`Init Duration`。
- 使用调用 ARN 为别名和调用的版本添加版本维度。如果您使用 Lambda 别名或版本来实现递增部署（如蓝-绿部署），则可以根据调用的别名查看指标。如果函数不使用别名或版本，则版本维度不适用。有关更多信息，请参阅 Lambda 函数别名。
- 向性能事件添加 `billed_mb_ms field` 以显示每次调用的成本。这些成本中未考虑与预置并发相关的任何成本。
- 向性能事件添加 `billed_duration` 和 `duration` 字段。

适用于版本 1.0.86.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:11`
美国东部（俄亥俄州）	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:11`
美国西部（加利福尼亚北部）	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:11`
美国西部（俄勒冈州）	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:11`
亚太地区（孟买）	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:11`
亚太地区（首尔）	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:11`
亚太地区（新加坡）	`arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:11`
亚太地区（悉尼）	`arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:11`

区域	ARN
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:11
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:11
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:11
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:11
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:11
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:11
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:11
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:11

1.0.54.0

1.0.54.0 版本是 Lambda Insights 扩展程序的初始版本。

适用于版本 1.0.54.0 的 ARN

下表列出了要在各个 AWS 区域中用于此版本扩展程序的适用 ARN。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:2

区域	ARN
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:2
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:2
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:2
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:2
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:2
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:2
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:2
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:2
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:2
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:2
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:2
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:2
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:2

区域	ARN
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:2
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:2

ARM64 平台

本节列出了适用于 ARM64 平台的 Lambda Insights 扩展程序版本，以及要在每个 AWS 区域用于这些扩展程序的 ARN。

> **Important**
>
> Lambda Insights 扩展版 1.0.317.0 和更高版本不支持 Amazon Linux 1。

1.0.333.0

版本 1.0.333.0 包括错误修复。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:22
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:18

区域	ARN
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:18
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:6
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:18
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:22
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:17
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:31
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20

区域	ARN
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:18
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:6
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:18
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:3
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:3

1.0.317.0

版本 1.0.317.0 包括取消对 Amazon Linux 1 平台的支持和错误修复。它还包括对 AWS GovCloud (US) 区域的支持。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:21

区域	ARN
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:17
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:17
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:5
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:17
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:16
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:30
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:17

区域	ARN
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:17
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:17
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:5
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:17
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:1
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:1

1.0.295.0

版本 1.0.295.0 包含对所有兼容的运行时系统的依赖项更新。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:16
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:16
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:4
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:16
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:15
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:17
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:18

区域	ARN
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:16
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:4
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:16
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16

1.0.275.0

版本 1.0.275.0 包含了对所有兼容的运行时的错误修复，以及对欧洲（西班牙）和亚太地区（海得拉巴）区域的支持。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:14
亚太地区（香港）	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:14
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:2
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:14
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:13
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:15

区域	ARN
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:14
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:14
欧洲（西班牙）	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:2
欧洲（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:14
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:14

1.0.273.0

版本 1.0.273.0 包含了对所有兼容运行时系统的错误修复。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:14
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:9
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:9
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:9
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:9
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:9
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:11
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:12

区域	ARN
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:9
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:10
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:9
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:10

1.0.229.0

版本 1.0.229.0 包含了对所有兼容运行时系统的错误修复。此外还增加了对以下区域的支持：

- 美国西部（加利福尼亚北部）
- 非洲（开普敦）
- Asia Pacific (Hong Kong)
- 亚太地区（雅加达）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 加拿大（中部）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲地区（斯德哥尔摩）
- 中东（巴林）
- 南美洲（圣保罗）

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:7
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:3
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
非洲（开普敦）	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:2
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:2
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:2

区域	ARN
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:7
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:2
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:4
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:11
加拿大（中部）	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:3
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
欧洲（西班牙）	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:2
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:3
欧洲地区（斯德哥尔摩）	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:3

区域	ARN
中东（巴林）	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:2
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:3

1.0.135.0

版本 1.0.135.0 包含针对 Lambda Insights 如何收集以及报告磁盘和文件描述符使用情况的错误修复。在此扩展程序的早期版本中，tmp_free 指标报告了函数运行时 /tmp 目录中的最大可用空间。此版本将指标更改为报告最小值，从而使其在评估磁盘使用情况时更有实用意义。有关 tmp 目录存储配额的更多信息，请参阅 Lambda 配额。

版本 1.0.135.0 现在还报告文件描述符使用情况（fd_use 和 fd_max）作为进程范围内的最大值，而不是报告操作系统级别。

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:2
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:2

区域	ARN
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2

1.0.119.0

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
亚太地区（孟买）	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
亚太地区（东京）	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1

区域	ARN
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1

使用控制台对现有 Lambda 函数启用 Lambda Insights

按照以下步骤，在 Lambda 控制台中对现有 Lambda 函数启用 Lambda Insights。

对 Lambda 函数启用 Lambda Insights

1. 通过 https://console.aws.amazon.com/lambda/ 打开 AWS Lambda 控制台。
2. 选择函数名称，然后选择以下屏幕中的 Configuration（配置）选项卡。
3. 在配置选项卡下，选择左侧导航菜单中的监控和操作工具，然后选择编辑。

   会将您定向到可以编辑监控工具的屏幕。
4. 在 Lambda Insights 增强监控旁，选择编辑。
5. 在 CloudWatch Lambda Insights 下，启用增强监控，然后选择保存。

使用 AWS CLI 对现有 Lambda 函数启用 Lambda Insights

按照以下步骤操作，使用 AWS CLI 对现有 Lambda 函数启用 Lambda Insights。

步骤 1：更新函数权限

更新函数权限

- 输入以下命令，将 CloudWatchLambdaInsightsExecutionRolePolicy 托管式 IAM 策略附加到函数的执行角色。

```
aws iam attach-role-policy \
--role-name function-execution-role \
```

```
--policy-arn "arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy"
```

步骤 2：安装 Lambda 扩展程序

输入以下命令安装 Lambda 扩展程序。将 `layers` 参数的 ARN 值替换为与您的区域和您要使用的扩展程序版本匹配的 ARN。有关更多信息，请参阅 Lambda Insights 扩展程序的可用版本。

```
aws lambda update-function-configuration \
   --function-name function-name \
   --layers "arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14"
```

步骤 3：启用 CloudWatch Logs VPC 终端节点

仅在函数运行于不具有 Internet 访问权限的私有子网中以及您尚未配置 CloudWatch Logs Virtual Private Cloud (VPC) 端点的情况下，才需要执行此步骤。

如果您需要执行此步骤，请输入以下命令，将占位符替换为 VPC 的信息。

有关更多信息，请参阅将 CloudWatch Logs 与接口 VPC 终端节点结合使用。

```
aws ec2 create-vpc-endpoint \
--vpc-id vpcId \
--vpc-endpoint-type Interface \
--service-name com.amazonaws.region.logs \
--subnet-id subnetId
--security-group-id securitygroupId
```

使用 AWS SAM CLI 对现有 Lambda 函数启用 Lambda Insights

按照以下步骤操作，使用 AWS SAM AWS CLI 对现有 Lambda 函数启用 Lambda Insights。

如果您尚未安装最新版本的 AWS SAM CLI，则必须先安装或升级到此版本。有关更多信息，请参阅安装 AWS SAM CLI。

步骤 1：安装层

要使 Lambda Insights 扩展程序可用于所有 Lambda 函数，请使用 Lambda Insights 层的 ARN 向 SAM 模板的 `Globals` 部分添加一个 `Layers` 属性。下方示例所示为将该层用于 Lambda Insights 的初始版本。有关 Lambda Insights 扩展程序层的最新版本，请参阅 Lambda Insights 扩展程序的可用版本。

```
Globals:
  Function:
    Layers:
      - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

若要仅为单个函数启用此层，请向该函数添加 Layers 属性，如本示例所示。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

步骤 2：添加托管式策略

对每个函数均需添加 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略。

AWS SAM 不支持全局策略，因此您必须对每个函数单独启用这些策略，如本示例所示。有关全局策略的更多信息，请参阅全局策略部分。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Policies:
        - CloudWatchLambdaInsightsExecutionRolePolicy
```

本地调用

AWS SAM CLI 支持 Lambda 扩展程序。但每个在本地执行的调用都会重置运行时环境。由于运行时在未发生关闭事件的情况下重新启动，Lambda Insights 数据将无法从本地调用中获得。有关更多信息，请参阅版本 1.6.0 - 添加对 AWS Lambda 扩展程序的本地测试支持。

故障排除

要排除 Lambda Insights 的安装问题，请将以下环境变量添加到 Lambda 函数中以启用调试日志记录。

```
Resources:
```

```
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          LAMBDA_INSIGHTS_LOG_LEVEL: info
```

使用 AWS CloudFormation 对现有 Lambda 函数启用 Lambda Insights

按照以下步骤操作，使用 AWS CloudFormation 对现有 Lambda 函数启用 Lambda Insights。

步骤 1：安装层

将 Lambda Insights 层添加到 Lambda Insights 层 ARN 中的 `Layers` 属性。下方示例所示为将该层用于 Lambda Insights 的初始版本。有关 Lambda Insights 扩展程序层的最新版本，请参阅 Lambda Insights 扩展程序的可用版本。

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

步骤 2：添加托管式策略

将 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略添加到您的函数执行角色。

```
Resources:
  MyFunctionExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy'
```

步骤 3：（可选）添加 VPC 终端节点

仅在函数运行于不具有 Internet 访问权限的私有子网中以及您尚未配置 CloudWatch Logs Virtual Private Cloud (VPC) 端点的情况下，才需要执行此步骤。有关更多信息，请参阅将 CloudWatch Logs 与接口 VPC 终端节点结合使用。

```
Resources:
  CloudWatchLogsVpcPrivateEndpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      PrivateDnsEnabled: 'true'
      VpcEndpointType: Interface
      VpcId: !Ref: VPC
      ServiceName: !Sub com.amazonaws.${AWS::Region}.logs
      SecurityGroupIds:
        - !Ref InterfaceVpcEndpointSecurityGroup
      SubnetIds:
        - !Ref PublicSubnet01
        - !Ref PublicSubnet02
        - !Ref PublicSubnet03
```

使用 AWS CDK 对现有 Lambda 函数启用 Lambda Insights

按照以下步骤操作，使用 AWS CDK 对现有 Lambda 函数启用 Lambda Insights。若要使用这些步骤，您必须已经使用 AWS CDK 来管理您的资源。

本部分的命令采用 TypeScript 语言。

首先，更新函数权限。

```
executionRole.addManagedPolicy(
 ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaInsightsExecutionRolePolicy')
);
```

接下来，在 Lambda 函数上安装扩展程序。将 `layerArn` 参数的 ARN 值替换为与您的区域和您要使用的扩展程序版本匹配的 ARN。有关更多信息，请参阅 Lambda Insights 扩展程序的可用版本。

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:580247275435:layer:LambdaInsightsExtension:14';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

如有必要，请为 CloudWatch Logs 启用 Virtual Private Cloud (VPC) 端点。仅在函数运行于不具有 Internet 访问权限的私有子网中以及您尚未配置 CloudWatch Logs VPC 终端节点的情况下，才需要执行此步骤。

```
const cloudWatchLogsEndpoint = vpc.addInterfaceEndpoint('cwl-gateway', {
```

```
  service: InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
});

cloudWatchLogsEndpoint.connections.allowDefaultPortFromAnyIpv4();
```

使用无服务器框架对现有 Lambda 函数启用 Lambda Insights

按照以下步骤使用无服务器框架对现有 Lambda 函数启用 Lambda Insights。有关无服务器框架的更多信息，请访问 serverless.com。

此启用方式是通过面向无服务器的 Lambda Insights 插件实现的。有关更多信息，请参阅 serverless-plugin-lambda-insights。

如果尚未安装最新版本的无服务器命令行界面，则必须先安装或升级到此版本。有关更多信息，请参阅开始使用无服务器框架开源和 AWS。

使用无服务器框架对 Lambda 函数上启用 Lambda Insights

1. 在无服务器目录中运行以下命令，为 Lambda Insights 安装无服务器插件：

   ```
   npm install --save-dev serverless-plugin-lambda-insights
   ```

2. 在 `serverless.yml` 文件中，将插件添加到 `plugins` 部分，如下所示：

   ```
   provider:
     name: aws
   plugins:
     - serverless-plugin-lambda-insights
   ```

3. 启用 Lambda Insights。
   - 您可以通过将以下属性添加到 serverless.yml 文件中，为每个函数单独启用 Lambda Insights

     ```
     functions:
       myLambdaFunction:
         handler: src/app/index.handler
         lambdaInsights: true #enables Lambda Insights for this function
     ```

   - 您可以通过在 `serverless.yml` 文件中添加以下自定义部分，为所有函数启用 Lambda Insights：

     ```
     custom:
     ```

```
  lambdaInsights:
    defaultLambdaInsights: true #enables Lambda Insights for all functions
```

4. 输入以下命令重新部署无服务器服务：

```
serverless deploy
```

此操作会重新部署所有函数，并为您指定的函数启用 Lambda Insights。
它通过添加 Lambda Insights 层以及使用 `arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy` IAM 策略附加必要的权限来启用 Lambda Insights。

对 Lambda 容器映像部署启用 Lambda Insights

要为部署为容器映像的 Lambda 函数启用 Lambda Insights，请在 Dockerfile 中添加相应的行。这些行会将 Lambda Insights 代理安装为容器镜像中的扩展程序。对于 x86-64 容器和 ARM64 容器，要添加的行是不同的。

> **Note**
>
> Lambda Insights 代理仅在使用 Amazon Linux 2 的 Lambda 运行时上受支持。

主题

- x86-64 容器映像部署
- ARM64 容器映像部署

x86-64 容器映像部署

对于在 x86-64 容器上运行并部署为容器映像的 Lambda 函数，要为其启用 Lambda Insights，请在 Dockerfile 中添加以下行。这些行会将 Lambda Insights 代理安装为容器镜像中的扩展程序。

```
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm
```

创建 Lambda 函数后，向函数的执行角色分配 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略，并对基于容器镜像的 Lambda 函数启用 Lambda Insights。

> **Note**
>
> 要使用较早版本的 Lambda Insights 扩展程序，请将先前命令中的 URL 替换为以下 URL：`https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.1.0.111.0.rpm`。目前，仅 1.0.111.0 及更高版本的 Lambda Insights 可用。有关更多信息，请参阅 Lambda Insights 扩展程序的可用版本。

验证 Linux 服务器上 Lambda Insights 代理软件包的签名

1. 输入以下命令下载公有密钥。

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
lambda-insights-extension.gpg
```

2. 输入以下命令将公有密钥导入到密钥环中。

```
shell$ gpg --import lambda-insights-extension.gpg
```

该输出值将类似于以下内容。记下 key 值，您将需要在下一步中使用该值。在此示例输出中，密钥值为 848ABDC8。

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1  (RSA: 1)
```

3. 输入以下命令验证指纹。将 key-value 替换为上一步中的密钥值。

```
shell$  gpg --fingerprint key-value
```

此命令输出中的指纹字符串应为 E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8。如果指纹字符串不匹配，请勿安装该代理，请联系 AWS。

4. 验证指纹后，您可以使用该指纹验证 Lambda Insights 代理软件包。输入以下命令下载软件包签名文件。

```
shell$  wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm.sig
```

5. 输入以下命令验证签名：

```
shell$ gpg --verify lambda-insights-extension.rpm.sig lambda-insights-extension.rpm
```

输出应与以下内容类似：

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

在预期输出中，可能会出现关于受信任签名的警告。只有当您或您信任的某个人对密钥进行了签名，密钥才是可信的。这并不意味着签名无效，只是您尚未验证公有密钥而已。

如果输出中包含 BAD `signature`，请检查是否正确执行了以上步骤。如果您继续获得 BAD `signature` 响应，请与 AWS 联系，并避免使用所下载的文件。

x86-64 示例

本部分包括一个对基于容器镜像的 Python Lambda 函数启用 Lambda Insights 的示例。

对 Lambda 容器镜像启用 Lambda Insights 的示例

1. 创建一个类似于以下的 Dockerfile：

```
FROM public.ecr.aws/lambda/python:3.8

// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 创建一个类似于以下内容的名为 index.py 的 Python 文件：

```
def handler(event, context):
  return {
    'message': 'Hello World!'
  }
```

3. 将 Dockerfile 和 index.py 放在同一目录中。然后，在该目录中执行以下步骤来构建 Docker 镜像并将其上载到 Amazon ECR。

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. 使用您刚刚创建的 Amazon ECR 镜像创建 Lambda 函数。

5. 将 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略分配给函数的执行角色。

ARM64 容器映像部署

对于在 AL2_aarch64 容器（采用 ARM64 架构）上运行并部署为容器映像的 Lambda 函数，要为其启用 Lambda Insights，请在 Dockerfile 中添加以下行。这些行会将 Lambda Insights 代理安装为容器镜像中的扩展程序。

```
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm
```

创建 Lambda 函数后，向函数的执行角色分配 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略，并对基于容器镜像的 Lambda 函数启用 Lambda Insights。

> **Note**
>
> 要使用较早版本的 Lambda Insights 扩展程序，请将先前命令中的 URL 替换为以下 URL：`https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.1.0.229.0.rpm`。目前，仅 1.0.229.0 及更高版本的 Lambda Insights 可用。有关更多信息，请参阅 Lambda Insights 扩展程序的可用版本。

验证 Linux 服务器上 Lambda Insights 代理软件包的签名

1. 输入以下命令下载公有密钥。

```
shell$ wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/lambda-insights-extension.gpg
```

2. 输入以下命令将公有密钥导入到密钥环中。

```
shell$ gpg --import lambda-insights-extension.gpg
```

该输出值将类似于以下内容。记下 key 值，您将需要在下一步中使用该值。在此示例输出中，密钥值为 848ABDC8。

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1  (RSA: 1)
```

3. 输入以下命令验证指纹。将 `key-value` 替换为上一步中的密钥值。

```
shell$  gpg --fingerprint key-value
```

此命令输出中的指纹字符串应为 `E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8`。如果指纹字符串不匹配，请勿安装该代理，请联系 AWS。

4. 验证指纹后，您可以使用该指纹验证 Lambda Insights 代理软件包。输入以下命令下载软件包签名文件。

```
shell$  wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm.sig
```

5. 输入以下命令验证签名：

```
shell$ gpg --verify lambda-insights-extension-arm64.rpm.sig lambda-insights-
extension-arm64.rpm
```

输出应与以下内容类似：

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

在预期输出中，可能会出现关于受信任签名的警告。只有当您或您信任的某个人对密钥进行了签名，密钥才是可信的。这并不意味着签名无效，只是您尚未验证公有密钥而已。

如果输出中包含 BAD `signature`，请检查是否正确执行了以上步骤。如果您继续获得 BAD `signature` 响应，请与 AWS 联系，并避免使用所下载的文件。

ARM64 示例

本部分包括一个对基于容器镜像的 Python Lambda 函数启用 Lambda Insights 的示例。

对 Lambda 容器镜像启用 Lambda Insights 的示例

1. 创建一个类似于以下的 Dockerfile：

```
FROM public.ecr.aws/lambda/python:3.8
// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 创建一个类似于以下内容的名为 `index.py` 的 Python 文件：

```
def handler(event, context):
  return {
```

```
    'message': 'Hello World!'
  }
```

3. 将 Dockerfile 和 `index.py` 放在同一目录中。然后，在该目录中执行以下步骤来构建 Docker 镜像并将其上载到 Amazon ECR。

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. 使用您刚刚创建的 Amazon ECR 镜像创建 Lambda 函数。
5. 将 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略分配给函数的执行角色。

更新函数的 Lambda Insights 扩展程序版本

作为最佳实践，建议您将 Lambda Insights 扩展程序更新到最新版本。本页中的主题说明如何操作。

> **Note**
>
> 本页说明如何更新已使用 Lambda Insights 的函数所使用的扩展程序版本。有关如何开始使用 Lambda Insights 的更多信息，请参阅 开始使用 Lambda Insights。

使用 Lambda 控制台更新 Lambda Insights 扩展程序版本

按照以下步骤，使用 Lambda 控制台更新 Lambda Insights 扩展程序版本。

使用 Lambda 控制台更新

1. 通过 https://console.aws.amazon.com/lambda/ 打开 AWS Lambda 控制台。
2. 选择函数的名称。

3. 在层部分中，选择编辑。
4. 在层列表中，搜索 LambdaInsightsExtension，然后将层版本更改为 Lambda Insights 扩展程序的可用版本 中列出的最新版本。
5. 选择保存。

使用 AWS CLI 更新 Lambda Insights 扩展程序版本

要使用 AWS CLI 更新 Lambda Insights 扩展程序版本，请输入以下命令。将 layers 参数的 ARN 值替换为与您的区域和您要使用的扩展程序版本匹配的 ARN。有关 Lambda Insights 扩展程序层最新版本的信息，请参阅 Lambda Insights 扩展程序的可用版本。

```
aws lambda update-function-configuration \
--function-name function-name \
--layers "arn:aws:lambda:us-west-1:111122223333:layer:LambdaInsightsExtension:53"
```

使用 AWS SAM CLI 更新一个或多个函数的 Lambda Insights 扩展程序

要更新所有 Lambda 函数的 Lambda Insights 扩展程序版本，请使用 Lambda Insights 层的 ARN 更新 AWS Serverless Application Model（SAM）模板 `Globals` 部分中的 `Layers` 属性。有关 Lambda Insights 扩展程序层最新版本的信息，请参阅 Lambda Insights 扩展程序的可用版本。

以下内容更新所有 Lambda 函数。

```
Globals:
  Function:
    Layers:
       - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

以下内容仅更新一个函数。

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

使用 AWS CloudFormation 更新一个或多个函数的 Lambda Insights 扩展程序

要使用 AWS CloudFormation 更新 Lambda Insights 扩展程序版本，请更新函数 AWS CloudFormation 资源内 `Layers` 属性中的扩展程序层，如以下示例所示。有关 Lambda Insights 扩展程序层最新版本的信息，请参阅 Lambda Insights 扩展程序的可用版本。

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

使用 AWS CDK 更新一个或多个函数的 Lambda Insights 扩展程序

通过将 `layerArn` 参数的 ARN 值替换为与您的区域和您要使用的扩展程序版本匹配的 ARN，您可以更新 Lambda 函数的扩展程序版本。有关 Lambda Insights 扩展程序层最新版本的信息，请参阅 Lambda Insights 扩展程序的可用版本。

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:111122223333:layer:LambdaInsightsExtension:53';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

使用无服务器框架更新一个或多个函数的 Lambda Insights 扩展程序

按照以下步骤使用无服务器框架更新现有 Lambda 函数的 Lambda Insights 扩展程序版本。有关无服务器框架的更多信息，请参阅无服务器框架文档。

此方法使用无服务器的 Lambda Insights 插件。有关更多信息，请参阅 serverless-plugin-lambda-insights。

如果尚未安装最新版本的无服务器命令行界面，则必须先安装或升级到此版本。有关更多信息，请参阅 Setting Up Serverless Framework With AWS。

使用 Lambda 控制台更新

1. 更新 Lambda Insights。如果您尚未执行此操作，则请在文件末尾添加 `custom` 部分，并在 `lambdaInsightsVersion` 属性中指定 Lambda Insights 版本。

```
custom:
    lambdaInsights:
        lambdaInsightsVersion: 53 #specify the Layer Version
```

2. 输入以下命令重新部署无服务器服务。

```
serverless deploy
```

对 Lambda 容器映像部署更新 Lambda Insights 扩展程序版本

要更新 Lambda 容器映像的 Lambda Insights，请按照 对 Lambda 容器映像部署启用 Lambda Insights 中的步骤使用最新版本的 Lambda Insights 重建映像。然后，使用 AWS CLI 更新函数代码并提供容器映像 URI 作为 `--image-uri` 参数的值。

查看 Lambda Insights 指标

在已调用的 Lambda 函数上安装 Lambda Insights 扩展程序后，您可以使用 CloudWatch 控制台查看您的指标。您可以查看关于多个函数的概览，或重点查看单个函数。

有关 Lambda Insights 指标的列表，请参阅 Lambda Insights 收集的指标。

查看 Lambda Insights 指标的多个函数概览

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，在 Lambda Insights 下，选择 Multi-function（多个函数）。

   页面顶部显示的图形包含已启用 Lambda Insights 的区域中所有 Lambda 函数的汇总指标。页面下部是一个列出了函数的表格。
3. 要按函数名称进行筛选以减少所显示的函数数量，请在页面顶部附近的框中键入函数的部分名称。
4. 要将此视图作为小组件添加到控制面板，请选择 Add to dashboard（添加到控制面板）。

查看单个函数的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，在 Lambda Insights 下，选择 Single-function（单个函数）。

   页面顶部显示了图形和所选函数的指标。

3. 如果启用了 X-Ray，则可以选择单个跟踪 ID。此操作会打开该调用的“X-Ray 跟踪地图”页面，您可以在此处使用缩小操作，以查看分布式跟踪以及涉及处理该特定事务的其他服务。有关 X-Ray 跟踪地图的更多信息，请参阅 Using the X-Ray Trace Map。
4. 要打开 CloudWatch Logs Insights 并放大特定错误，请选择页面底部表格的 View logs（查看日志）。
5. 要将此视图作为小组件添加到控制面板，请选择 Add to dashboard（添加到控制面板）。

与 Application Insights 集成

Amazon CloudWatch Application Insights 可帮助您监控应用程序，并在应用程序资源和技术堆栈中指定和设置关键指标、日志和告警。有关更多信息，请参阅 Amazon CloudWatch Application Insights。

您可以启用 Application Insights，从 Lambda 函数中收集其他数据。如果您尚未启用此功能，可以通过在 Lambda Insights 控制面板中的性能视图下方，选择 Application Insights 中的 Auto-configure Application Insights（自动配置 Application Insights）来启用它。

如果您已设置 CloudWatch Application Insights 来监控 Lambda 函数，则 Application Insights 控制面板将显示在 Application Insights 选项卡中的 Lambda Insights 控制面板下方。

Lambda Insights 收集的指标

Lambda Insights 会从其安装位置的 Lambda 函数中收集多个指标。其中一些指标可作为 CloudWatch 指标中的时间序列聚合数据提供。其他指标不会聚合到时间序列数据中，但可以使用 CloudWatch Logs Insights 在嵌入式指标格式日志条目中找到。

以下指标可作为 `LambdaInsights` 命名空间中 CloudWatch 指标中的时间序列聚合数据提供。

指标名称	Dimensions	描述
`cpu_total_time`	function_name function_name、版本	`cpu_system_time` 和 `cpu_user_time` 的总和。 单位：毫秒
`init_duration`	function_name function_name、版本	在 Lambda 执行环境生命周期的 `init` 阶段中花费的时间。

指标名称	Dimensions	描述
		单位：毫秒
memory_utilization	function_name function_name、版本	最大内存（以分配给函数的内存的百分比表示）。 单位：百分比
rx_bytes	function_name function_name、版本	函数接收的字节数。 单位：字节
tmp_used		/tmp 目录中已使用的空间量。 单位：字节
tx_bytes	function_name function_name、版本	函数发送的字节数。 单位：字节
total_memory	function_name function_name、版本	分配给您的 Lambda 函数的内存量。此内存量与函数的内存大小相同。 单位：兆字节
total_network	function_name function_name、版本	rx_bytes 和 tx_bytes 的总和。由于 Lambda 运行时做出了网络调用，即使对于不执行输入/输出任务的函数，此值通常也大于零。 单位：字节

指标名称	Dimensions	描述
used_memory_max	function_name function_name、版本	函数沙盒的测量内存。 单位：兆字节

以下指标可以使用 CloudWatch Logs Insights 在嵌入式指标格式日志条目中找到。有关 CloudWatch Logs Insights 的更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。

有关嵌入式指标格式的更多信息，请参阅在日志中嵌入指标。

指标名称	描述
cpu_system_time	CPU 执行内核代码所花费的时间。 单位：毫秒
cpu_total_time	cpu_system_time 和 cpu_user_time 的总和。 单位：毫秒
cpu_user_time	CPU 执行用户代码所花费的时间。 单位：毫秒
fd_max	可用的最大文件描述符数。 单位：计数
fd_use	正在使用的最大文件描述符数。 单位：计数
memory_utilization	最大内存（以分配给函数的内存的百分比表示）。 单位：百分比
rx_bytes	函数接收的字节数。

指标名称	描述	
	单位：字节	
tx_bytes	函数发送的字节数。 单位：字节	
threads_max	函数进程正在使用的线程数。作为函数作者，您无法控制运行时创建的线程的初始数量。 单位：计数	
tmp_max	/tmp 目录中可用的空间量。 单位：字节	
total_memory	分配给您的 Lambda 函数的内存量。此内存量与函数的内存大小相同。 单位：兆字节	
total_network	rx_bytes 和 tx_bytes 的总和。由于 Lambda 运行时做出了网络调用，即使对于不执行输入/输出任务的函数，此值通常也大于零。 单位：字节	
used_memory_max	函数沙盒的测量内存。 单位：字节	

问题排查和已知问题

对任何问题执行问题排查的第一步是在 Lambda Insights 扩展程序中启用调试日志记录。为此，请对 Lambda 函数设置以下环境变量：LAMBDA_INSIGHTS_LOG_LEVEL=info。有关更多信息，请参阅使用 AWS Lambda 环境变量。

Lambda Insights 扩展程序会将日志发送到与函数相同的日志组中 (/aws/lambda/*function-name*)。查看这些日志，看看错误是否可能与设置问题有关。

我没有看到 Lambda Insights 中的任何指标

如果您未看到预计可以看到的 Lambda Insights 指标，请检查以下可能性：

- 指标可能只是存在延迟：如果函数尚未被调用或数据尚未刷新，则您将看不到 CloudWatch 中的指标。有关更多信息，请参阅本部分后文的已知问题。
- 确认 Lambda 函数具有正确的权限：确保已将 CloudWatchLambdaInsightsExecutionRolePolicy IAM 策略分配给了函数的执行角色。
- 检查 Lambda 运行时：Lambda Insights 仅支持某些 Lambda 运行时。有关支持的运行时的列表，请参阅 Lambda Insights。

  例如，要在 Java 8 上使用 Lambda Insights，您必须使用 `java8.al2` 运行时，而不能使用 `java8` 运行时。
- 检查网络访问：Lambda 函数可能使用的是不具有 Internet 访问权限的 VPC 私有子网，并且您没有为 CloudWatch Logs 配置 VPC 终端节点。为了帮助调试此问题，您可以设置环境变量 `LAMBDA_INSIGHTS_LOG_LEVEL=info`。

已知问题

数据延迟可能高达 20 分钟。当函数处理程序完成时，Lambda 会冻结沙盒，此操作也会冻结 Lambda Insights 扩展程序。当函数运行时，我们使用基于函数 TPS 的自适应批处理策略来输出数据。但是，如果长时间停止调用函数，并且缓冲区中仍有事件数据，则这些数据可能会被延迟到直到 Lambda 关闭空闲沙盒。当 Lambda 关闭沙盒时，我们会刷新缓冲的数据。

示例遥测事件

每次调用启用了 Lambda Insights 的 Lambda 函数，都会向 `/aws/lambda-insights` 日志组写入一个日志事件。每个日志事件都包含嵌入式指标格式的指标。有关嵌入式指标格式的更多信息，请参阅在日志中嵌入指标。

您可以使用以下方法分析这些日志事件：

- CloudWatch 控制台的 Lambda Insights 部分（见 查看 Lambda Insights 指标 中的说明）。
- 使用 CloudWatch Logs Insights 记录事件查询。有关更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。
- `LambdaInsights` 命名空间中收集的指标，您可以通过使用 CloudWatch 指标来绘制这些指标的图形。

以下是嵌入式指标格式的 Lambda Insights 日志事件示例。

```
{
    "_aws": {
        "Timestamp": 1605034324256,
        "CloudWatchMetrics": [
            {
                "Namespace": "LambdaInsights",
                "Dimensions": [
                    [ "function_name" ],
                    [ "function_name", "version" ]
                ],
                "Metrics": [
                    { "Name": "memory_utilization", "Unit": "Percent" },
                    { "Name": "total_memory", "Unit": "Megabytes" },
                    { "Name": "used_memory_max", "Unit": "Megabytes" },
                    { "Name": "cpu_total_time", "Unit": "Milliseconds" },
                    { "Name": "tx_bytes", "Unit": "Bytes" },
                    { "Name": "rx_bytes", "Unit": "Bytes" },
                    { "Name": "total_network", "Unit": "Bytes" },
                    { "Name": "init_duration", "Unit": "Milliseconds" }
                ]
            }
        ],
        "LambdaInsights": {
            "ShareTelemetry": true
        }
    },
    "event_type": "performance",
    "function_name": "cpu-intensive",
    "version": "Blue",
    "request_id": "12345678-8bcc-42f7-b1de-123456789012",
    "trace_id": "1-5faae118-12345678901234567890",
    "duration": 45191,
    "billed_duration": 45200,
    "billed_mb_ms": 11571200,
    "cold_start": true,
    "init_duration": 130,
    "tmp_free": 538329088,
    "tmp_max": 551346176,
    "threads_max": 11,
    "used_memory_max": 63,
    "total_memory": 256,
    "memory_utilization": 24,
```

```
    "cpu_user_time": 6640,
    "cpu_system_time": 50,
    "cpu_total_time": 6690,
    "fd_use": 416,
    "fd_max": 32642,
    "tx_bytes": 4434,
    "rx_bytes": 6911,
    "timeout": true,
    "shutdown_reason": "Timeout",
    "total_network": 11345,
    "agent_version": "1.0.72.0",
    "agent_memory_avg": 10,
    "agent_memory_max": 10
}
```

使用 Contributor Insights 分析高基数数据

您可以使用 Contributor Insights 分析日志数据，并创建显示贡献者数据的时间序列。您可以查看有关前 N 个贡献者、独特贡献者总数及其使用情况的指标。这有助于您找到排名靠前的说话者，并了解影响系统性能的人员或内容。例如，您可以查找无效主机，确定最多的网络用户或查找产生最多错误的 URL。

您可以从 Scratch 构建规则，并且在使用 AWS Management Console 时，还可以使用 AWS 已创建的示例规则。规则定义要用于定义贡献者的日志字段，例如 `IpAddress`。还可以筛选日志数据来查找和分析各个贡献者的行为。

CloudWatch 还提供了内置规则，可用于分析来自其他 AWS 服务的指标。

所有规则都将实时分析传入数据。

如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则您可以在该监控账户中创建 Contributor Insights 规则，以分析源账户和该监控账户中的日志组。您还可以创建单个规则来分析多个账户中的日志组。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

使用 Contributor Insights，需要为每个与规则匹配的日志事件付费。有关更多信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> 只有当规则引用的数字值介于 -1e9 和 1e9 之间时，Contributor Insights 才会匹配日志条目。如果日志条目中的值超出此范围，Contributor Insights 会跳过该日志条目。

主题

- 创建 Contributor Insights 规则
- Contributor Insights 规则语法
- Contributor Insights 规则示例
- 查看 Contributor Insights 报告
- 绘制规则生成的指标的图表
- 使用 Contributor Insights 内置规则

创建 Contributor Insights 规则

您可创建规则以分析日志数据。可以评估任何采用 JSON 或常用日志格式 (CLF) 的日志。这包括遵循这些格式之一的自定义日志以及来自 AWS 服务的日志，例如 Amazon VPC 流日志、Amazon Route 53 DNS 查询日志、Amazon ECS 容器日志以及来自 AWS CloudTrail、Amazon SageMaker、Amazon RDS、AWS AppSync 和 API Gateway 的日志。

在规则中指定字段名称或值时，所有匹配都区分大小写。

在创建规则时，您可以使用内置示例规则，也可以从 Scratch 创建自己的规则。Contributor Insights 包括以下日志类型的示例规则：

- Amazon API Gateway 日志
- Amazon Route 53 公有 DNS 查询日志
- Amazon Route 53 resolver 查询日志
- CloudWatch Container Insights 日志
- VPC 流日志

如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则除了为该监控账户中的日志组创建规则外，您可以为与该监控账户关联的源账户中的日志组创建 Contributor Insights 规则。您还可以设置单个规则来监控不同账户中的日志组。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

> **Important**
>
> 当您向用户授予 `cloudwatch:PutInsightRule` 权限时，默认情况下，该用户可以创建一个规则来评估 CloudWatch Logs 中的任何日志组。您可以添加 IAM 策略条件，以限制用户的

这些权限，使其包含和排除特定的日志组。有关更多信息，请参阅 使用条件键限制 Contributor Insights 用户对日志组的访问。

要使用内置示例规则创建规则，请执行以下操作：

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，依次选择 Insights、Contributor Insights。
3. 选择创建规则。
4. 对于 Select log group(s)（选择日志组），请选择您希望您的规则监控的日志组。您可以选择多达 20 个日志组。如果您登录的账户是为 CloudWatch 跨账户可观测性设置的监控账户，则可以选择源账户中的日志组，也可以设置单个规则来分析不同账户中的日志组。
   - （可选）要选择名称以特定字符串开头的所有日志组，请选择 Select by prefix match（按前缀匹配选择）下拉菜单，然后输入前缀。如果这是监控账户，则可以选择要在其中进行搜索的账户，否则将选择所有账户。

   > **Note**
   >
   > 您需要为与您的规则匹配的每个日志事件支付费用。如果您选择 Select by prefix match（按前缀匹配选择）下拉菜单，请注意前缀可以匹配多少个日志组。如果您搜索了超出预期数量的日志组，可能会产生意外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

5. 对于 Rule type（规则类型），选择 Sample rule（示例规则）。然后选择 Select sample rule（选择示例规则），并选择相应的规则。
6. 示例规则已填写 Log format（日志格式）、Contribution（贡献）、Filters（筛选条件）和 Aggregate on（聚合）字段。您可以根据需要对这些值进行调整。
7. 选择下一步。
8. 为 Rule name (规则名称) 输入一个名称。有效字符包括 A-Z、a-z、0-9、（连字符）、（下划线）和（半角句点）。
9. 选择是创建处于已禁用状态还是已启用状态的规则。如果您选择启用规则，它将立即开始对您的数据进行分析。运行启用的规则时，您需要支付费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

Contributor Insights 仅在创建规则后分析新的日志事件。规则无法处理之前已由 CloudWatch Logs 处理的日志事件。

10. （可选）对于 Tags（标签），请添加一个或多个键/值对作为此规则的标签。标签可帮助您识别和组织 AWS 资源并跟踪 AWS 成本。有关更多信息，请参阅 标记 Amazon CloudWatch 资源。
11. 选择创建。

要从 Scratch 创建规则，请执行以下操作：

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Contributor Insights。
3. 选择创建规则。
4. 对于 Select log group(s)（选择日志组），请选择您希望您的规则监控的日志组。您可以选择多达 20 个日志组。如果您登录的账户是为 CloudWatch 跨账户可观测性设置的监控账户，则可以选择源账户中的日志组，也可以设置单个规则来分析不同账户中的日志组。

    - （可选）要选择名称以特定字符串开头的所有日志组，请选择 Select by prefix match（按前缀匹配选择）下拉菜单，然后输入前缀。

    > **Note**
    >
    > 您需要为与您的规则匹配的每个日志事件支付费用。如果您选择 Select by prefix match（按前缀匹配选择）下拉菜单，请注意前缀可以匹配多少个日志组。如果您搜索了超出预期数量的日志组，可能会产生意外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

5. 对于 Rule type（规则类型），请选择 Custom rule（自定义规则）。
6. 对于 Log format (日志格式)，请选择 JSON 或 CLF。
7. 您可以通过以下方式完成规则的创建过程：使用向导，或选择 Syntax (语法) 选项卡并手动指定规则语法。

    要继续使用此向导，请执行以下操作：

    a. 对于 Contribution (贡献)、Key (键)，请输入要报告的贡献者类型。此报告将显示该贡献者类型的前 N 个值。

有效条目为任何具有值的日志字段。示例包括 **`requestId`**、**`sourceIPaddress`** 和 **`containerID`**。

有关查找特定日志组中日志的日志字段名称的信息，请参阅查找日志字段。

大于 1 KB 的键将被截断至 1KB。

b. （可选）选择 Add new key（添加新键）以添加更多键。可以在规则中包含最多四个键。如果输入多个键，则报告中的贡献者将由键的唯一值组合定义。例如，如果您指定三个键，则三个键的每个唯一值组合将被计为一个独特贡献者。

c. （可选）如果要添加筛选条件以缩小结果范围，请选择 Add filter（添加筛选条件）。对于 Match（匹配），输入要作为筛选条件的日志字段的名称。对于 Condition（条件），选择比较运算符，并输入要作为筛选条件的值。

您可以在规则中最多添加 4 个筛选条件。多个筛选条件通过 AND 逻辑联接，因此，仅返回与所有筛选条件都匹配的日志事件。

> **Note**
>
> 遵循比较运算符的数组（例如 `In`、`NotIn` 或者 `StartsWith`）可以包含最多 10 个字符串值。有关 Contributor Insights 规则语法的更多信息，请参阅 Contributor Insights 规则语法。

d. 对于 Aggregate on（聚合），选择 Count（计数）或 Sum（总计）。选择 Count（计数）会使贡献者排名基于出现次数。选择 Sum（总计）会使排名基于您为 Contribution（贡献）、Value（值）指定的字段值的总和。

8. 要输入规则作为 JSON 对象而不是使用此向导，请执行以下操作：

a. 选择 Syntax (语法) 选项卡。

b. 在 Rule body (规则正文) 中，输入规则的 JSON 对象。有关规则语法的信息，请参阅 Contributor Insights 规则语法。

9. 选择下一步。

10. 为 Rule name (规则名称) 输入一个名称。有效字符为 A-Z、a-z、0-9、“-”、“_”和“.”。

11. 选择是创建处于已禁用状态还是已启用状态的规则。如果您选择启用规则，它将立即开始对您的数据进行分析。运行启用的规则时，您需要支付费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

Contributor Insights 仅在创建规则后分析新的日志事件。规则无法处理之前已由 CloudWatch Logs 处理的日志事件。

12. （可选）对于 Tags（标签），请添加一个或多个键/值对作为此规则的标签。标签可帮助您识别和组织 AWS 资源并跟踪 AWS 成本。有关更多信息，请参阅 标记 Amazon CloudWatch 资源。
13. 选择下一步。
14. 确认您输入的设置，然后选择 Create rule（创建规则）。

可以禁用、启用或删除已创建的规则。

在 Contributor Insights 中启用、禁用或删除规则

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Contributor Insights。
3. 在规则列表中，选中单个规则旁边的复选框。

   内置规则由 AWS 服务创建，无法编辑、禁用或删除这些规则。
4. 选择 Actions (操作)，然后选择所需选项。

查找日志字段

创建规则时，您需要知道日志组的日志条目中的字段名称。

查找日志组中的日志字段

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中的 Logs (日志) 下，选择 Insights (见解)。
3. 在查询编辑器上方，选择要查询的一个或多个日志组。

   当您选择日志组时，CloudWatch Logs Insights 会自动检测日志组数据中的字段，并将其显示在右侧窗格中的 Discovered fields（发现的字段）中。

Contributor Insights 规则语法

此部分说明了 Contributor Insights 规则的语法。仅当您通过输入 JSON 块来创建规则时使用此语法。如果您使用此向导创建规则，则无需知道语法。有关使用此向导创建规则的更多信息，请参阅 创建 Contributor Insights 规则。

日志事件字段名称和值的所有规则匹配都区分大小写。

以下示例说明了 JSON 日志的语法。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*",
        "Log-group-name2"
    ],
    "LogFormat": "JSON",
    "Contribution": {
        "Keys": [
            "$.ip"
        ],
        "ValueOf": "$.requestBytes",
        "Filters": [
            {
                "Match": "$.httpMethod",
                "In": [
                    "PUT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Contributor Insights 规则中的字段

架构

用于分析 CloudWatch Logs 数据 Schema 的规则的值必须始终为 {"Name": "CloudWatchLogRule", "Version": 1}

LogGroupNames

字符串数组。对于数组中的每个元素，您可以选择在字符串末尾使用 * 以包含名称以该前缀开头的所有日志组。

将通配符用于日志组名称时要小心。您需要为与规则匹配的每个日志事件支付费用。如果不小心搜索了超出预期数量的日志组，可能会产生意外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

LogGroupARN

如果您在 CloudWatch 跨账户可观测性监控账户中创建此规则，则可以使用 LogGroupARNs 在与该监控账户关联的源账户中指定日志组，然后指定该监控账户本身的日志组。您必须在规则中指定 LogGroupNames 或 LogGroupARNs，但不能同时指定两者。

LogGroupARNs 是一个字符串数组。对于该数组中的每个元素，在某些情况下可以选择使用 * 作为通配符。例如，您可以指定 arn:aws:logs:us-west-1:*:log-group/MyLogGroupName2，以便在美国西部（北加利福尼亚）地区的所有源账户和该监控账户中指定名为 MyLogGroupName2 的日志组。您还可以指定 arn:aws:logs:us-west-1:111122223333:log-group/GroupNamePrefix*，以便在 111122223333 中指定美国西部（北加利福尼亚）所有名称以 GroupNamePrefix 开头的日志组。

您不能将部分 AWS 账户 ID 指定为带通配符的前缀。

将通配符用于日志组 ARN 时要小心。您需要为与规则匹配的每个日志事件支付费用。如果不小心搜索了超出预期数量的日志组，可能会产生意外费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

LogFormat

有效值为 JSON 和 CLF。

贡献

此对象包含一个具有最多个四个成员的 Keys 数组、（可选）一个 ValueOf、（可选）一个具有最多四个 Filters 的数组。

键

一个最多具有四个日志字段的数组，这些字段用作维度以对贡献者进行分类。如果您输入多个键，则这些键的每个唯一值组合将被计为一个独特贡献者。必须使用 JSON 属性格式表示法指定字段。

ValueOf

（可选）仅当您指定 Sum 作为 AggregateOn 的值时指定此选项。ValueOf 指定具有数值的日志字段。在此类规则中，贡献者按其在此字段中的值的和来进行排名，而不是按其在日志条目中的出现次数进行排名。例如，如果您希望按贡献者在一段时间内的 BytesSent 总数对其进行排名，则需要将 ValueOf 设置为 BytesSent，并为 AggregateOn 指定 Sum。

筛选条件

（可选）指定具有最多四个筛选条件的数组可缩小报告中包含的日志事件的范围。如果您指定多个筛选条件，则 Contributor Insights 会使用逻辑 AND 运算符对它们进行评估。您可以使用此选项来筛掉搜索中不相关的日志事件，也可以使用它选择单个贡献者来分析其行为。

数组中的每个成员必须包含一个 Match 字段和一个指示要使用的匹配运算符类型的字段。

Match 字段指定要在筛选条件中评估的日志字段。使用 JSON 属性格式表示法指定日志字段。

匹配的运算符字段必须是下列项目之一：In、NotIn、StartsWith、GreaterThan、LessThan、EqualTo、NotEqualTo 或 IsPresent。如果运算符字段为 In、NotIn 或 StartsWith，则它后跟一个要检查的字符串值数组。Contributor Insights 使用 OR 运算符评估字符串值的数组。该数组可以包含最多 10 个字符串值。

如果运算符字段为 GreaterThan、LessThan、EqualTo 或 NotEqualTo，则它后跟要比较的单个数值。

如果运算符字段为 IsPresent，则它后跟 true 或 false。此运算符将根据日志事件中是否存在指定的日志字段来匹配日志事件。isPresent 仅使用 JSON 属性的叶节点中的值。例如，查找 c-count 的匹配项的筛选条件不会返回具有值 details.c-count.c1 的日志事件。

有关四个筛选条件示例，请参阅以下内容:

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThan": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

有效值为 Count 和 Sum。指定是基于出现次数，还是基于 ValueOf 字段中指定的字段值的总和来聚合报告。

JSON 属性格式表示法

Keys、ValueOf 和 Match 字段遵循带点符号的 JSON 属性格式，其中 $ 表示 JSON 对象的根。这之后是句点，然后是带子属性名称的字母数字字符串。支持多个属性级别。

字符串的第一个字符只能是 A-Z 或 a-z。字符串的后续字符可以是 A-Z、a-z 或 0-9。

以下列表显示了 JSON 属性格式的有效示例：

```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

CLF 日志规则中的其他字段

常用日志格式 (CLF) 日志事件不像 JSON 那样具有字段的名称。要提供用于 Contributor Insights 规则的字段，CLF 日志事件可被视为具有从 1 开始的索引的数组。您可以将第一个字段指定为 **"1"**，将第二个字段指定为 **"2"**，以此类推。

要使 CLF 日志的规则更易于读取，可以使用 Fields。这可让您为 CLF 字段位置提供命名别名。例如，您可以指定位置“4”为 IP 地址。指定后，IpAddress 可用作规则中的 Keys、ValueOf 和 Filters 中的属性。

以下是使用 Fields 字段的 CLF 日志规则的示例。

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "IpAddress",
        "7": "StatusCode"
    },
    "Contribution": {
        "Keys": [
            "IpAddress"
```

```
        ],
        "Filters": [
            {
                "Match": "StatusCode",
                "EqualTo": 200
            }
        ]
    },
    "AggregateOn": "Count"
}
```

Contributor Insights 规则示例

此部分包含的示例说明了 Contributor Insights 规则的使用案例。

VPC 流日志：按源和目标 IP 地址进行的字节传输

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "srcaddr",
        "5": "dstaddr",
        "10": "bytes"
    },
    "Contribution": {
        "Keys": [
            "srcaddr",
            "dstaddr"
        ],
        "ValueOf": "bytes",
        "Filters": []
    },
    "AggregateOn": "Sum"
}
```

VPC 流日志：最大 HTTPS 请求数

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "5": "destination address",
        "7": "destination port",
        "9": "packet count"
    },
    "Contribution": {
        "Keys": [
            "destination address"
        ],
        "ValueOf": "packet count",
        "Filters": [
            {
                "Match": "destination port",
                "EqualTo": 443
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

VPC 流日志：被拒绝的 TCP 连接数

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
```

```
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Route 53 NXDomain 按源地址响应

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.rcode",
                "StartsWith": [
                    "NXDOMAIN"
                ]
            }
        ],
        "Keys": [
            "$.srcaddr"
```

```
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 Resolver 按域名查询

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_name"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 Resolver 按查询类型和源地址查询

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_type",
            "$.srcaddr"
        ]
    },
```

```
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

查看 Contributor Insights 报告

要查看报告数据的图表和您的规则找到的贡献者排名列表，请执行以下步骤。

查看规则报告

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Contributor Insights。
3. 在该规则列表中，选择规则的名称。

   图表显示了过去三小时内的规则结果。图表下方的表显示了前 10 名贡献者。
4. 要更改表中显示的贡献者数，请选择图表顶部的 Top 10 contributors (前 10 名贡献者)。
5. 要筛选图表以仅显示单个贡献者的结果，请在表图例中选择该贡献者。要再次显示所有贡献者，请在图例中再次选择同一个贡献者。
6. 要更改报告中显示的时间范围，请选择图表顶部的 15m（15 分钟）、30m（30 分钟）、1h（1 小时）、2h（2 小时）、3h（3 小时）或 custom（自定义）。

   报告的最大时间范围为 24 小时，但您可以选择最多 15 天前发生的 24 小时时段。要选择过去的时段，请选择 custom (自定义)、absolute (绝对)，然后指定您的时段。
7. 要更改用于聚合和贡献者排名的时段的长度，请选择图表顶部的 period (周期)。查看较长的时段通常会显示更平滑的报告，其中包含的峰值很少。选择较短的时段更有可能显示峰值。
8. 要将此图表添加到 CloudWatch 控制面板，请选择 Add to dashboard（添加到控制面板）。
9. 要打开 CloudWatch Logs Insights 查询窗口（此报告中的日志组已加载到查询框中），请选择 View logs（查看日志）。
10. 要将报告数据导出到剪贴板或 CSV 文件，请选择 Export (导出)。

绘制规则生成的指标的图表

Contributor Insights 提供了一个指标数学函数，即 INSIGHT_RULE_METRIC。您可以使用此函数将 Contributor Insights 报告中的数据添加到 CloudWatch 控制台的 Metrics（指标）选项卡中的图表。您也可以根据此数学函数来设置警报。有关指标数学函数的更多信息，请参阅 使用指标数学。

要使用此指标数学函数，您必须登录到同时具有 cloudwatch:GetMetricData 和 cloudwatch:GetInsightRuleReport 权限的账户。

语法为 INSIGHT_RULE_METRIC(*ruleName*, *metricName*)。*ruleName* 是 Contributor Insights 规则的名称，*metricName* 是以下列表中的值之一。*metricName* 的值决定了数学函数返回的数据类型。

- UniqueContributors – 每个数据点的独特贡献者数。
- MaxContributorValue – 每个数据点的顶级贡献者的值。对于图表中的每个数据点，确定的贡献者可能会变化。

  如果此规则按 Count 进行聚合，则每个数据点的顶级贡献者是该时段内出现次数最多的贡献者。如果此规则按 Sum 进行聚合，则排在最前面的贡献者是此时段内具有该规则的 Value 指定的日志字段中最大总和的贡献者。
- SampleCount – 规则匹配的数据点数。
- Sum – 该数据点表示的时段内来自所有贡献者的值的总和。
- Minimum – 该数据点所表示的时段内单个观察结果中的最小值。
- Maximum – 该数据点所表示的时段内单个观察结果中的最大值。
- Average – 该数据点表示的时段内来自所有贡献者的平均值。

为 Contributor Insights 指标数据设置告警

您可以使用函数 INSIGHT_RULE_METRIC 对 Contributor Insights 生成的指标设置告警。例如，您可以根据已被拒绝的传输控制协议（TCP）连接的百分比来创建告警。要开始使用这种类型的告警，您可以创建类似以下两个示例中所示的规则:

示例规则: "RejectedConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
```

```
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

示例规则: "TotalConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
```

```
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [{
            "Match": "protocol",
            "EqualTo": 6
        }],
        "AggregateOn": "Sum"
    }
}
```

创建规则后，您可以选择 CloudWatch 控制台中的 Metrics（指标）选项卡，在其中使用以下示例指标数学表达式来绘制 Contributor Insights 报告的数据的图表:

示例: 指标数学表达式

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

在该示例中，指标数学表达式 e3 返回所有已被拒绝的 TCP 连接。如果您想要在已被拒绝的 TCP 连接达到 20% 时收到通知，则可以修改表达式，将阈值从 100 更改为 20。

> **Note**
>
> 在 Metrics（指标）部分中，您可以对您监控的指标设置告警。在 Graphed metrics（绘制的指标）选项卡中，您可以选择 Actions（操作）列下的 Create alarm（创建告警）。Create alarm（创建告警）图标看起来像个铃铛。

有关绘制指标的图表和使用指标数学函数的更多信息，请参阅以下部分: 向 CloudWatch 图表中添加数学表达式。

使用 Contributor Insights 内置规则

您可以使用 Contributor Insights 内置规则来分析其他 AWS 服务的指标。以下服务支持内置规则：

- Amazon DynamoDB 开发人员指南中的 Amazon DynamoDB 的 Contributor Insights。
- AWS PrivateLink 指南中的使用内置的 Contributor Insights 规则。

Amazon CloudWatch Application Insights

Amazon CloudWatch Application Insights 简化了观察您的应用程序和基础 AWS 资源的过程。它可用帮助您为应用程序资源设置最佳的监控，以持续分析数据，以便找出应用程序出现问题的迹象。Application Insights 由 Sagemaker 和其他 AWS 技术提供支持，它提供了自动化控制面板以显示监控的应用程序的潜在问题，从而帮助您快速找出应用程序和基础设施当前存在的问题。Application Insights 提高了应用程序运行状况的可见性，可以帮助您缩短平均修复时间 (MTTR)，以便对应用程序进行故障排除。

在将应用程序添加到 Amazon CloudWatch Application Insights 时，它扫描应用程序中的资源，并在 CloudWatch 上为应用程序组件建议和配置指标和日志。示例应用程序组件可能包括 SQL Server 后端数据库和 Microsoft IIS/Web 层。Application Insights 使用历史数据分析指标模式以检测异常情况，并从应用程序、操作系统和基础设施日志中持续检测错误和异常情况。它使用分类算法和内置规则组合以关联这些观察结果。然后，它自动创建控制面板以显示相关的观察结果和问题严重性信息，以帮助您确定操作的优先级。对于 .NET 和 SQL 应用程序堆栈中的常见问题（例如，应用程序延迟、SQL Server 备份失败、内存泄漏、过大的 HTTP 请求以及输入/输出操作取消），它提供了额外的洞察，以指出可能的根本原因和解决步骤。与 AWS SSM OpsCenter 进行内置集成后，您可以通过运行相关的 Systems Manager Automation 文档来解决问题。

Sections

- Amazon CloudWatch Application Insights 是什么？
- Amazon CloudWatch Application Insights 的工作方式
- 开始使用 Amazon CloudWatch Application Insights
- Application Insights 跨账户可观测性
- 使用组件配置
- 使用 CloudFormation 模板创建和配置 CloudWatch Application Insights 监控
- 教程：为 SAP ASE 设置监控
- 教程：为 SAP HANA 设置监控

- 教程：为 SAP NetWeaver 设置监控
- 查看和排查 Amazon CloudWatch Application Insights 检测到的问题
- Amazon CloudWatch Application Insights 支持的日志和指标

Amazon CloudWatch Application Insights 是什么？

CloudWatch Application Insights 可帮助您监控使用 Amazon EC2 实例以及其他应用程序资源的应用程序。它可在应用程序资源和技术堆栈（例如，Microsoft SQL Server 数据库、Web (IIS) 和应用程序服务器、操作系统、负载均衡器和队列）中识别和设置关键指标、日志和警报。它会持续监控指标和日志，以检测异常情况和错误并将它们关联起来。在检测到错误和异常情况时，Application Insights 生成 CloudWatch Events，您可以使用这些事件来设置通知或执行操作。为了帮助进行故障排除，它会为检测到的问题创建自动化控制面板，其中包含关联的指标异常情况和日志错误，以及可指出潜在根本原因的其他洞察。自动化控制面板可帮助您快速采取修复操作，以保持应用程序正常运行，并防止对应用程序的终端用户造成影响。此外，它还会创建 OpsItems，以便您可以使用 AWS SSM OpsCenter 解决问题。

您可以在 CloudWatch 上配置重要的计数器，例如镜像写入事务数/秒、恢复队列长度、事务延迟以及 Windows 事件日志。当 SQL HA 工作负载发生故障转移事件或问题（例如，查询目标数据库的访问权限受限）时，CloudWatch Application Insights 会提供自动化洞察。

CloudWatch Application Insights 与 AWS Launch Wizard 集成，从而为在 AWS 上部署 SQL Server HA 工作负载提供一键式监控设置体验。当您在 Launch Wizard 控制台上选择使用 Application Insights 设置监控和洞察的选项时，CloudWatch Application Insights 会自动在 CloudWatch 上设置相关指标、日志和告警，并开始监控新部署的工作负载。您可以在 CloudWatch 控制台上查看自动化的洞察和检测到的问题，以及 SQL Server HA 工作负载的运行状况。

内容

- 功能
- 概念
- 定价
- 相关服务
- 支持的应用程序组件
- 支持的技术堆栈

功能

Application Insights 提供以下功能。

自动为应用程序资源设置监视器

CloudWatch Application Insights 可减少为应用程序设置监控所需的时间。它扫描应用程序资源，提供可自定义的建议指标和日志列表，并在 CloudWatch 上设置，从而为所需的应用程序资源（例如 Amazon EC2 和 Elastic Load Balancer (ELB)）提供可见性，从而减少为应用程序设置监控所需的时间。它还会为监控的指标设置动态警报。将根据前两周检测到的异常情况自动更新警报。

问题检测和通知

CloudWatch Application Insights 检测应用程序出现潜在问题的迹象，例如指标异常情况和日志错误。它关联这些观察结果以找出应用程序的潜在问题。然后，它生成 CloudWatch Events，可以配置这些事件以接收通知或执行操作。这样，您无需为指标或日志错误创建单独的警报。

故障排除

CloudWatch Application Insights 为检测到的问题创建 CloudWatch 自动化控制面板。这些控制面板显示有关问题的详细信息（包括关联的指标异常情况和日志错误）以帮助您进行故障排除。它们还提供额外的信息，以指出异常情况和错误的潜在根本原因。

概念

要了解 Application Insights 如何监控应用程序，以下概念是非常重要的。

组件

对组成应用程序的类似资源进行自动分组、单独分组或自定义分组。我们建议将类似资源分组到自定义组件中，以更好地进行监控。

观察

为应用程序或应用程序资源检测到的单个事件（指标异常情况、日志错误或异常）。

问题

关联、分类和分组相关的观察结果以检测问题。

有关 CloudWatch Application Insights 的其他重要概念的定义，请参阅 Amazon CloudWatch 概念。

定价

CloudWatch Application Insights 使用 CloudWatch Metrics、Logs 和 Events 为选定的应用程序资源设置建议的指标和日志，以便就检测到的问题发出通知。将根据 CloudWatch 定价对您的 AWS 账户收取这些功能的费用。对于检测到的问题，Application Insights 还会创建 SSM OpsItems，以通知您相关问题。此外，Application Insights 会创建 SSM Parameter Store 参数，以便在您的实例上配置 CloudWatch 代理。Amazon EC2 Systems Manager 的功能根据 SSM 定价收费。不会向您收取设置帮助、监控、数据分析或问题检测费用。

CloudWatch Application Insights 的成本

Amazon EC2 的成本包括使用以下功能：

- CloudWatch 代理
  - CloudWatch 代理日志组
  - CloudWatch 代理指标
  - Prometheus 日志组（用于 JMX 工作负载）

所有资源的成本包括使用以下功能：

- CloudWatch 告警（大部分成本）
- SSM OpsItems（最低成本）

成本计算示例

本示例中的成本是根据以下方案考虑的。

您创建了一个包含以下内容的资源组：

- 安装了 SQL Server 的 Amazon EC2 实例。
- 附加的 Amazon EBS 卷。

当您使用 CloudWatch Application Insights 载入此资源组时，会检测到安装在 Amazon EC2 实例上的 SQL Server 工作负载。CloudWatch Application Insights 开始监控以下指标。

为 SQL Server 实例监控以下指标：

- CPUUtilization

- StatusCheckFailed
- Memory % Committed Bytes in Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec
- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections
- SQLServer:Locks Number of Deadlocks/sec
- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

为附加到 SQL Server 实例的卷监控以下指标：

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumeIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

此方案的成本是根据 CloudWatch 定价页面和 SSM 定价页面计算的：

- 自定义指标

对于此方案，上述指标中的 13 个指标是使用 CloudWatch 代理发送到 CloudWatch 的。这些指标被视为自定义指标。每个自定义指标的成本为每月 0.3 美元。这些自定义指标的总成本为 13 * 0.3 美元 = 3.90 美元/月。

- 警报

  对于此方案，CloudWatch Application Insights 总共监控 26 个指标，这会创建 26 个告警。每个告警的成本为每月 0.1 美元。告警的总成本为 26 * 0.1 美元 = 2.60 美元/月。

- 数据摄取和错误日志

  数据摄取的成本为 0.05 美元/GB，SQL Server 错误日志的存储成本为 0.03 美元/GB。数据摄取和错误日志的总成本为 0.05 美元/GB + 0.03 美元/GB = 0.08 美元/GB。

- Amazon EC2 Systems Manager OpsItems

  系统为 CloudWatch Application Insights 检测到的每个问题创建一个 SSM OpsItem。对于应用程序中的 n 个问题，总成本为 0.00267 * n 美元/月。

相关服务

以下服务与 CloudWatch Application Insights 一起使用：

相关 AWS 服务

- Amazon CloudWatch 在系统范围内提供资源使用率、应用程序性能和运行状况信息。它收集并跟踪指标，发送警报通知，根据您定义的规则自动更新监控的资源，并允许您监控自己的自定义指标。CloudWatch Application Insights 是通过 CloudWatch 启动的，具体来说，是在 CloudWatch 默认运行控制面板中启动的。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- CloudWatch Container Insights 从容器化应用程序和微服务中收集、聚合及汇总指标与日志。您可以使用 Container Insights 来监控 Amazon ECS、Amazon Elastic Kubernetes Service 以及 Amazon EC2 上的 Kubernetes 平台。在 Container Insights 或 Application Insights 控制台中启用 Application Insights 后，Application Insights 会将检测到的问题显示在 Container Insights 控制面板上。有关更多信息，请参阅Container Insights。
- Amazon DynamoDB 是完全托管式 NoSQL 数据库服务器，您可以用它免除操作和扩展分布式数据库的管理工作负担，因而无需担心硬件预置、设置和配置、复制、软件修补或集群扩展等问题。此外，DynamoDB 提供了加密静态，这可以消除在保护敏感数据时涉及的操作负担和复杂性。
- Amazon EC2 在 AWS 云中提供可扩展的计算容量。您可以使用 Amazon EC2 启动所需数量的虚拟服务器，配置安全性和联网，以及管理存储。您可以扩展或缩减以处理需求变化或使用高峰，从而

减少预测流量的需求。有关更多信息，请参阅适用于 Linux 实例的 Amazon EC2 用户指南 或适用于 Windows 实例的 Amazon EC2 指南。

- Amazon Elastic Block Store (Amazon EBS) 提供了数据块级的存储卷以用于 Amazon EC2 实例。Amazon EBS 卷的行为类似于原始、未格式化的块储存设备。您可以将这些卷作为设备挂载在实例上。附加到实例的 Amazon EBS 卷公开为独立于实例生命周期而持续存在的存储卷。您可以在这些卷上创建文件系统，或者以使用块储存设备（如硬盘）的任何方式使用这些卷。您可以动态更改附加到实例的卷的配置。有关更多信息，请参阅 Amazon EBS 用户指南。
- Amazon EC2 Auto Scaling 帮助确保您具有正确数量的 EC2 实例以处理应用程序负载。有关更多信息，请参阅 Amazon EC2 Auto Scaling 用户指南。
- Elastic Load Balancing 在多个可用区中的多个目标（如 EC2 实例、容器和 IP 地址）之间分配传入的应用程序或网络流量。有关更多信息，请参阅 Elastic Load Balancing 用户指南。
- IAM 是一项 Web 服务，可帮助您安全地控制用户对 AWS 资源的访问权限。可以通过 IAM 控制哪些用户可以使用您的 AWS 资源（身份验证），以及控制他们可以使用的资源和使用资源的方式（授权）。有关更多信息，请参阅 Amazon CloudWatch 的身份验证和访问控制。
- 您可以通过 AWS Lambda 构建由事件触发的函数组成的无服务器应用程序，并使用 CodePipeline 和 AWS CodeBuild 自动部署这些应用程序。有关更多信息，请参阅 AWS Lambda 应用程序。
- AWS Launch Wizard for SQL Server 缩短了将 SQL Server 高可用性解决方案部署到云所需的时间。您可以在服务控制台上输入应用程序要求，包括性能、节点数量和连接性，并且，AWS Launch Wizard 可标识正确的 AWS 资源来部署和运行 SQL Server Always On 应用程序。
- AWS Resource Groups 帮助您划分组成应用程序的资源。通过使用 Resource Groups，您可以同时管理和自动完成针对大量资源的任务。只能为单个应用程序注册一个资源组。有关更多信息，请参阅 AWS Resource Groups 用户指南。
- Amazon SQS 提供了一个安全、持久且可用的托管队列，以允许您集成和分离分布式软件系统和组件。有关更多信息，请参阅 Amazon SQS 用户指南。
- AWS Step Functions 是一个无服务器函数构建工具，它允许您对各种 AWS 服务和资源（包括 AWS Lambda 函数）排序为结构化的可视化工作流。有关更多信息，请参阅《AWS Step Functions 用户指南》。
- AWS SSM OpsCenter 跨服务聚合并标准化 OpsItem，同时提供关于每个 OpsItem、相关 OpsItem 和相关资源的上下文调查数据。OpsCenter 还提供 Systems Manager Automation 文档 (runbook)，让您可以快速解决问题。您可以为每个 OpsItem 指定可搜索的自定义数据。您还可以按状态和源查看自动生成的 OpsItem 相关摘要报告。有关更多信息，请参阅《AWS Systems Manager 用户指南》。

- Amazon API Gateway 是一项 AWS 服务，用于创建、发布、维护、监控和保护任意规模的 REST、HTTP 和 WebSocket API。API 开发人员可以创建能够访问 AWS 或其他 Web 服务以及存储在 AWS 云中的数据的 API。有关更多信息，请参阅 Amazon API Gateway 用户指南。

  > **Note**
  >
  > Application Insights 仅支持 REST API 协议（v1 的 API Gateway 服务）。

- Amazon Elastic Container Service (Amazon ECS) 是一项完全托管式的容器编排服务。您可以使用 Amazon ECS 运行最敏感和任务关键型的应用程序。有关更多信息，请参阅 Amazon Elastic Container Service 开发人员指南。
- Amazon Elastic Kubernetes Service (Amazon EKS) 是一项托管式服务，可让您在 AWS 上轻松运行 Kubernetes，而无需安装、操作或维护您自己的 Kubernetes 控制面板或节点。Kubernetes 是一个用于实现容器化应用程序的部署、扩缩和管理自动化的开源系统。有关更多信息，请参阅 Amazon EKS 用户指南。
- Amazon EC2 上的 Kubernetes。Kubernetes 是一个开源软件，可帮助您大规模部署和管理容器化应用程序。Kubernetes 可管理 Amazon EC2 计算实例的集群，并通过部署、维护和扩展流程在这些实例上运行容器。使用 Kubernetes，您可以在本地和云中使用相同的工具集运行任何类型的容器化应用程序。有关更多信息，请参阅 Kubernetes 文档：入门。
- Amazon FSx可帮助您启动和运行流行的 AWS 完全托管式文件系统。使用 Amazon FSx，您可以利用通用开源文件系统和获商业许可的文件系统的功能集和性能来避免耗时的管理任务。有关更多信息，请参阅 Amazon FSx 文档。
- Amazon Simple Notification Service (SNS) 是一项完全托管式消息收发服务，适用于应用程序间和应用程序与人之间的通信。您可以将 Amazon SNS 配置为由 Application Insights 对其进行监控。当将 Amazon SNS 配置为监控资源时，Application Insights 会跟踪 SNS 指标，帮助确定 SNS 消息可能会遇到问题或失败的原因。
- Amazon Elastic File System（Amazon EFS）是一种完全托管式的弹性 NFS 文件系统，可用于 AWS Cloud 服务和本地资源。它专为在不中断应用程序的情况下按需扩展到 PB 级容量而打造。文件系统会在您添加和移除文件时自动增大和缩小容量，无需预调配和管理容量来满足容量增长需求。有关更多信息，请参阅 Amazon Elastic File System 文档。

相关第三方服务

- 对于在 Application Insights 中监控的某些工作负载和应用程序，Prometheus JMX Exporter 使用 AWS Systems Manager Distributor 安装，以便 CloudWatch Application Insights 可以检索 Java 特

定的指标。当您选择监控 Java 应用程序时，Application Insights 会自动为您安装 Prometheus JMX Exporter。

支持的应用程序组件

CloudWatch Application Insights 可扫描您的资源组以识别应用程序组件。组件可能采用单独分组、自动分组（例如自动扩缩组中的实例或负载均衡器后面的实例）或自定义分组（将各个 Amazon EC2 实例分组在一起）。

CloudWatch Application Insights 支持以下组件：

AWS 组件

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing：Application Load Balancer 和经典负载均衡器（指定和配置这些负载均衡器的所有目标实例）。
- Amazon EC2 Auto Scaling 组：AWS Auto Scaling（为所有目标实例动态配置自动扩缩组；如果应用程序纵向扩展，CloudWatch Application Insights 将自动配置新实例）。基于 CloudFormation 堆栈的 Resource Groups 不支持自动扩缩组。
- AWS Lambda
- Amazon Simple Queue Service(Amazon SQS)
- Amazon DynamoDB 表
- Amazon S3 存储桶指标
- AWS Step Functions
- Amazon API Gateway REST API 阶段
- Amazon Elastic Container Service (Amazon ECS)：集群、服务和任务
- Amazon Elastic Kubernetes Service (Amazon EKS)：集群
- Amazon EC2 上的 Kubernetes：在 EC2 上运行的 Kubernetes 集群
- Amazon SNS 主题

CloudWatch Application Insights 当前不会追踪任何其他组件类型资源。如果在 Application Insights 应用程序中未显示某种支持的组件类型，则您拥有并由 Application Insights 监控的其他应用程序可能已注册和管理该组件。

支持的技术堆栈

您可以在“Application tier（应用程序层）”下拉菜单选项中选择以下技术之一，从而使用 CloudWatch Application Insights 监控在 Windows Server 和 Linux 操作系统上运行的应用程序：

- 前端：Microsoft Internet 信息服务 (IIS) Web 服务器
- 工件层：
  - NET Framework。
  - .NET 内核
- 应用程序：
  - Java
  - SAP NetWeaver 标准、分布式和高可用性部署
- Active Directory
- SharePoint
- 数据库：
  - 在 Amazon RDS 或 Amazon EC2 上运行的 Microsoft SQL Server（包括 SQL Server 高可用性配置，请参阅 组件配置示例）。
  - 在 Amazon RDS、Amazon Aurora 或 Amazon EC2 上运行的 MySQL
  - 在 Amazon RDS 或 Amazon EC2 上运行的 PostgreSQL
  - Amazon DynamoDB 表
  - 在 Amazon RDS 或 Amazon EC2 上运行的 Oracle
  - 单个 Amazon EC2 实例和多个 EC2 实例上的 SAP HANA 数据库
  - 跨可用区 SAP HANA 数据库高可用性设置
  - 单个 Amazon EC2 实例上的 SAP Sybase ASE 数据库
  - 跨可用区 SAP Sybase ASE 数据库高可用性设置

如果上面列出的任何技术堆栈都不适用于您的应用程序资源，则您可以通过在 Manage monitoring（管理监控）页面上的“Application tier（应用程序层）”下拉菜单中选择 Custom（自定义）来监控您的应用程序堆栈。

Amazon CloudWatch Application Insights 的工作方式

此部分包含有关 CloudWatch Application Insights 的工作方式的信息，其中包括：

- Application Insights 监控应用程序的方式
- 数据留存
- 配额
- CloudWatch Application Insights 使用的 AWS Systems Manager (SSM) 软件包
- CloudWatch Application Insights 使用的 AWS Systems Manager（SSM）文档

Application Insights 监控应用程序的方式

Application Insights 监控应用程序的方式如下。

应用程序发现和配置

首次将应用程序添加到 CloudWatch Application Insights 时，它扫描应用程序组件以建议关键指标、日志和其他数据源，以便监控您的应用程序。然后，您可以根据这些建议配置应用程序。

数据预处理

CloudWatch Application Insights 持续分析在应用程序资源中监控的数据源，以查找指标异常情况和日志错误（观察结果）。

智能问题检测

CloudWatch Application Insights 引擎使用分类算法和内置规则关联观察结果，以检测应用程序中的问题。为了帮助进行故障排除，它创建一些自动化 CloudWatch 控制面板，其中包含有关问题的上下文信息。

警报和操作

在 CloudWatch Application Insights 检测到应用程序出现问题时，它生成 CloudWatch Events 以通知您该问题。有关如何设置这些事件的更多信息，请参阅检测到的问题的 Application CloudWatch Events 和通知。

示例方案

您具有一个由 SQL Server 数据库支持的 ASP .NET 应用程序。突然，您的数据库由于内存压力过高而开始发生故障。这会导致应用程序性能下降，并且可能会导致在 Web 服务器和负载均衡器中出现 HTTP 500 错误。

借助于 CloudWatch Application Insights 及其智能分析，您可以检查动态创建的控制面板（显示相关指标和日志文件片段）以找出导致问题的应用程序层。在这种情况下，问题可能出在 SQL 数据库层。

数据留存

CloudWatch Application Insights 将问题保留 55 天，并将观察结果保留 60 天。

配额

有关 CloudWatch Application Insights 的默认配额，请参阅 Amazon CloudWatch Application Insights 端点和配额。除非另有说明，否则，每个配额针对的是每个 AWS 区域。如需请求增加服务配额，请联系 AWS Support。许多服务包含无法更改的配额。有关特定服务的配额的更多信息，请参阅针对该服务的文档。

CloudWatch Application Insights 使用的 AWS Systems Manager (SSM) 软件包

本节中列出的软件包由 Application Insights 使用，可独立管理并可通过 AWS Systems Manager Distributor 部署。有关 SSM Distributor 的更多信息，请参阅 AWS Systems Manager 用户指南中的 AWS Systems Manager Distributor。

软件包：

- AWSObservabilityExporter-JMXExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure
- AWSObservabilityExporter-HAClusterExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure
- AWSObservabilityExporter-SQLExporterInstallAndConfigure

`AWSObservabilityExporter-JMXExporterInstallAndConfigure`

您可以从 Prometheus JMX Exporter 中检索工作负载特定的 Java 指标，以供 Application Insights 配置和监控告警。在 Application Insights 控制台的 Manage monitoring（管理监控）页面上，在 Application tier（应用程序层）下拉菜单选择 JAVA application（JAVA 应用程序）。然后在 JAVA Prometheus exporter configuration（JAVA Prometheus Exporter 配置）下，选择您的 Collection method（收集方法）和 JMX port number（JMX 端口号）。

要使用 AWS Systems Manager Distributor 来打包、安装和配置 AWS 提供的 Prometheus JMX Exporter（独立于 Application Insights），请完成以下步骤。

使用 Prometheus JMX Exporter SSM 软件包的先决条件

- 2.3.1550.0 版或更高版本的 SSM Agent 已安装

- JAVA_HOME 环境变量已设置

安装和配置 **AWSObservabilityExporter-JMXExporterInstallAndConfigure** 软件包

AWSObservabilityExporter-JMXExporterInstallAndConfigure 软件包是一个 SSM Distributor 软件包，您可以使用它来安装和配置 Prometheus JMX Exporter。当 Prometheus JMX Exporter 发送 Java 指标时，可以配置 CloudWatch 代理来检索 CloudWatch 服务的指标。

1. 根据您的首选项，准备位于 Prometheus GitHub 存储库中的 Prometheus JMX Exporter YAML 配置文件。使用示例配置和选项说明为您提供指导。
2. 将编码为 Base64 的 Prometheus JMX Exporter YAML 配置文件复制到 SSM Parameter Store 中的新 SSM 参数。
3. 导航到 SSM Distributor 控制台，然后打开 Owned by Amazon（Amazon 所拥有）选项卡。选择 AWSObservabilityExporter-JMXExporterInstallAndConfigure，然后选择 Install one time（安装一次）。
4. 通过将“Additional Arguments（其他实际参数）”替换为以下内容来更新您在第一步中创建的 SSM 参数：

```
{
  "SSM_EXPORTER_CONFIGURATION": "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_EXPOSITION_PORT": "9404"
}
```

> **Note**
>
> 端口 9404 是用于发送 Prometheus JMX 指标的默认端口。您可以更新此端口。

示例：配置 CloudWatch 代理以检索 Java 指标

1. 如前面的步骤所述，安装 Prometheus JMX Exporter。然后检查端口状态，确认它是否已正确安装在您的实例上。

   在 Windows 实例示例上的成功安装

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)
StatusCode : 200
StatusDescription : OK
```

```
Content : # HELP jvm_info JVM version info
```

在 Linux 实例示例上的成功安装

```
$ curl localhost:9404
# HELP jmx_config_reload_failure_total Number of times configuration have failed to
 be reloaded.
# TYPE jmx_config_reload_failure_total counter
jmx_config_reload_failure_total 0.0
```

2. 创建 Prometheus 服务发现 YAML 文件。以下示例服务发现文件将执行以下操作：

   - 将 Prometheus JMX Exporter 主机端口指定为 `localhost: 9404`。
   - 将标签（`Application`、`ComponentName` 和 `InstanceId`）附加到指标，这些指标可以设置为 CloudWatch 指标维度。

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn:aws:elasticloadbalancing:us-
east-1:123456789012:loadbalancer/app/sampl-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. 创建 Prometheus JMX Exporter 配置 YAML 文件。以下示例配置文件可指定以下内容：

   - 指标检索任务间隔和超时周期。
   - 指标检索任务（`jmx` 和 `sap`）也称为抓取，其中包括任务名称、一次返回的最大时间序列以及服务发现文件路径。

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
```

```
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. 验证 CloudWatch 代理是否已安装在您的 Amazon EC2 实例上，以及版本是否为 1.247346.1b249759 或更高版本。要在 EC2 实例上安装 CloudWatch 代理，请参阅安装 CloudWatch 代理。要验证版本，请参阅查找有关 CloudWatch 代理版本的信息。
5. 配置 CloudWatch 代理。有关如何配置 CloudWatch 代理配置文件的更多信息，请参阅手动创建或编辑 CloudWatch 代理配置文件。以下示例 CloudWatch 代理配置文件可执行以下操作：

   - 指定 Prometheus JMX Exporter 配置文件路径。
   - 指定要向其发布 EMF 指标日志的目标日志组。
   - 为每个指标名称指定两组维度。
   - 发送 8 个（每个指标名称 2 组维度，每组 4 个指标名称）CloudWatch 指标。

```
{
   "logs":{
      "logs_collected":{
         ....
      },
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-test-cluster",
            "log_group_name":"prometheus-test",
            "prometheus_config_path":"/tmp/prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent",
               "metric_unit":{
                  "jvm_threads_current":"Count",
                  "jvm_gc_collection_seconds_sum":"Second",
                  "jvm_memory_bytes_used":"Bytes"
               },
               "metric_declaration":[
                  {
                     "source_labels":[
                        "job"
                     ],
                     "label_matcher":"^jmx$",
```

```
                    "dimensions":[
                       [
                          "InstanceId",
                          "ComponentName"
                       ],
                       [
                          "ComponentName"
                       ]
                    ],
                    "metric_selectors":[
                       "^java_lang_threading_threadcount$",
                       "^java_lang_memory_heapmemoryusage_used$",
                       "^java_lang_memory_heapmemoryusage_committed$"
                    ]
                 }
              ]
           }
        }
     },
     "metrics":{
        ....
     }
}
```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure

您可以从 Prometheus HANA Database Exporter 中检索工作负载特定的 SAP HANA 指标，以供 Application Insights 配置和监控告警。有关更多信息，请参阅本指南中的设置 SAP HANA 数据库以进行监控。

要使用 AWS Systems Manager Distributor 来打包、安装和配置 AWS 提供的 Prometheus HANA Database Exporter 软件包（独立于 Application Insights），请完成以下步骤。

使用 Prometheus HANA Database Exporter SSM 软件包的先决条件

- 2.3.1550.0 版或更高版本的 SSM Agent 已安装
- SAP HANA 数据库
- Linux 操作系统（SUSE Linux、RedHat Linux）
- 带有 SAP HANA 数据库监控凭证的密钥（使用 AWS Secrets Manager）。使用键/值对格式创建密钥，指定密钥用户名，然后输入值的数据库用户。再添加一个密钥密码，然后输入值的密码。有关如

何创建密钥的更多信息，请参阅 AWS Secrets Manager 用户指南中的创建密钥。密钥必须采用以下格式：

```
{
  "username": "<database_user>",
  "password": "<database_password>"
}
```

安装和配置 AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure 软件包

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure 软件包是一个 SSM Distributor 软件包，您可以使用它来安装和配置 Prometheus HANA Database Exporter。当 Prometheus HANA Database Exporter 发送 HANA 数据库指标时，就可以配置 CloudWatch 代理来检索 CloudWatch 服务的指标。

1. 在 SSM Parameter Store 中创建 SSM 参数以存储该 Exporter 的配置。以下为示例参数值。

```
{\"exposition_port\":9668,\"multi_tenant\":true,\"timeout\":600,\"hana\":{\"host\":
\"localhost\",\"port\":30013,\"aws_secret_name\":\"HANA_DB_CREDS\",\"scale_out_mode
\":true}}
```

> **Note**
>
> 在此示例中，导出任务仅在具有活跃 SYSTEM 数据库的 Amazon EC2 实例上运行，并在其他 EC2 实例上保持空闲状态，以免指标重复。该 Exporter 可以检索 SYSTEM 数据库中的所有数据库租户信息。

2. 在 SSM Parameter Store 中创建 SSM 参数以存储该 Exporter 的指标查询。软件包可以接受多个指标参数。每个参数必须具有有效的 JSON 对象格式。以下为示例参数值：

```
{\"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE,
 SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION,
 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait
 Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0,
 TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0))
 IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor'
 GROUP BY HOST, MEASURED_ELEMENT_NAME;\":{\"enabled\":true,\"metrics\":[{\"name\":
\"hanadb_cpu_user\",\"description\":\"Percentage of CPU time spent by HANA DB in user
```

```
 space, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"USER_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_system
\",\"description\":\"Percentage of CPU time spent by HANA DB in Kernel space,
 over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"SYSTEM_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_waitio
\",\"description\":\"Percentage of CPU time spent by HANA DB in IO mode, over the
 last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":\"WAITIO_PCT\",
\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_busy\",\"description
\":\"Percentage of CPU time spent by HANA DB, over the last minute (in seconds)\",
\"labels\":[\"HOST\",\"CORE\"],\"value\":\"BUSY_PCT\",\"unit\":\"percent\",\"type\":
\"gauge\"},{\"name\":\"hanadb_cpu_idle\",\"description\":\"Percentage of CPU time not
 spent by HANA DB, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE
\"],\"value\":\"IDLE_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"}]}}
```

有关指标查询的更多信息，请参阅 GitHub 上的 SUSE / hanadb_exporter 存储库。

3. 导航到 SSM Distributor 控制台，然后打开 Owned by Amazon（Amazon 所拥有）选项卡。依次选择 AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure* 和 Install one time（一次性安装）。
4. 通过将“Additional Arguments（其他实际参数）”替换为以下内容来更新您在第一步中创建的 SSM 参数：

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_EXPORTER_METRICS_1": "{{ssm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}}",
  "SSM_EXPORTER_METRICS_2": "{{ssm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}}"
}
```

5. 选择带有 SAP HANA 数据库的 Amazon EC2 实例，然后选择 Run（运行）。

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure

您可以从 Prometheus HANA Cluster Exporter 中检索工作负载特定的高可用性 (HA) 集群指标，以供 Application Insights 配置和监控 SAP HANA 高可用性设置的告警。有关更多信息，请参阅本指南中的设置 SAP HANA 数据库以进行监控。

要使用 AWS Systems Manager Distributor 来打包、安装和配置 AWS 提供的 Prometheus HA Cluster Exporter 软件包（独立于 Application Insights），请完成以下步骤。

使用 Prometheus HA Cluster Exporter SSM 软件包的先决条件

- 2.3.1550.0 版或更高版本的 SSM Agent 已安装
- 适用于 Pacemaker、Corosync、SBD 和 DRBD 的 HA 集群
- Linux 操作系统（SUSE Linux、RedHat Linux）

安装和配置 **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure** 软件包

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure 软件包是一个 SSM Distributor 软件包，您可以使用它来安装和配置 Prometheus HA Cluster Exporter。当 Prometheus HANA Database Exporter 发送集群指标时，就可以配置 CloudWatch 代理来检索 CloudWatch 服务的指标。

1. 在 SSM Parameter Store 中创建 SSM 参数，以 JSON 格式存储该 Exporter 的配置。以下为示例参数值。

   ```
   {\"port\":\"9664\",\"address\":\"0.0.0.0\",\"log-level\":\"info\",\"crm-mon-path
   \":\"/usr/sbin/crm_mon\",\"cibadmin-path\":\"/usr/sbin/cibadmin\",\"corosync-
   cfgtoolpath-path\":\"/usr/sbin/corosync-cfgtool\",\"corosync-quorumtool-path\":\"/
   usr/sbin/corosync-quorumtool\",\"sbd-path\":\"/usr/sbin/sbd\",\"sbd-config-path\":
   \"/etc/sysconfig/sbd\",\"drbdsetup-path\":\"/sbin/drbdsetup\",\"enable-timestamps
   \":false}
   ```

   有关该 Exporter 配置的更多信息，请参阅 GitHub 上的 ClusterLabs / ha_cluster_exporter 存储库。

2. 导航到 SSM Distributor 控制台，然后打开 Owned by Amazon（Amazon 所拥有）选项卡。依次选择 AWSObservabilityExporter-HAClusterExporterInstallAndConfigure* 和 Install one time（一次性安装）。

3. 通过将“Additional Arguments（其他实际参数）”替换为以下内容来更新您在第一步中创建的 SSM 参数：

   ```
   {
     "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"
   }
   ```

4. 选择带有 SAP HANA 数据库的 Amazon EC2 实例，然后选择 Run（运行）。

AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure

您可以从 Prometheus SAP 主机导出器中检索特定于工作负载的 SAP NetWeaver 指标，以便 Application Insights 为 SAP NetWeaver 分布式和高可用性部署配置和监控告警。有关更多信息，请参阅 开始使用 Amazon CloudWatch Application Insights。

要使用 AWS Systems Manager Distributor 来打包、安装和配置 SAP 主机导出器软件包（独立于 Application Insights），请完成以下步骤。

使用 Prometheus SAP 主机导出器 SSM 软件包的先决条件

- 2.3.1550.0 版或更高版本的 SSM Agent 已安装
- SAP NetWeaver 应用程序服务器
- Linux 操作系统（SUSE Linux、RedHat Linux）

安装和配置 AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure 软件包

AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure 软件包是一个 SSM Distributor 软件包，您可以使用它来安装和配置 SAP NetWeaver Prometheus 指标导出器。当 Prometheus 导出器发送 SAP NetWeaver 指标时，可以配置 CloudWatch 代理来检索 CloudWatch 服务的指标。

1. 在 SSM Parameter Store 中创建 SSM 参数，以 JSON 格式存储该 Exporter 的配置。以下为示例参数值。

```
{\"address\":\"0.0.0.0\",\"port\":\"9680\",\"log-level\":\"info\",\"is-HA\":false}
```

   - address

     发送 Prometheus 指标的目标地址。默认值为 localhost。

   - port (远程调试端口)

     发送 Prometheus 指标的目标端口。默认值为 9680。

   - is-HA

     对于 SAP NetWeaver 高可用性部署，其值为 true。对于所有其他部署，其值为 false。

2. 导航到 SSM Distributor 控制台，然后打开 Owned by Amazon（Amazon 所拥有）选项卡。选择 AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure，然后选择 Install one time（安装一次）。
3. 通过将“Additional Arguments（其他实际参数）”替换为以下内容来更新您在第一步中创建的 SSM 参数：

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_INSTANCES_NUM": "<instances_number seperated by comma>"
}
```

示例

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:exporter_config_paramter}}",
  "SSM_INSTANCES_NUM": "11,12,10",
  "SSM_SID": "PR1"
}
```

4. 选择带有 SAP NetWeaver 应用程序的 Amazon EC2 实例，然后选择 Run（运行）。

> **Note**
>
> Prometheus 导出器在本地端点上维护 SAP NetWeaver 指标。只有 Amazon EC2 实例上的操作系统用户才能访问本地端点。因此，在安装导出器软件包后，所有操作系统用户都可以使用这些指标。默认的本地端点是 `localhost:9680/metrics`。

`AWSObservabilityExporter-SQLExporterInstallAndConfigure`

您可以从 Prometheus SQL 导出器中检索工作负载特定的 SQL Server 指标，以供 Application Insights 监控关键指标。

要使用 AWS Systems Manager Distributor 来打包、安装和配置 SQL 导出器软件包（独立于 Application Insights），请完成以下步骤。

使用 Prometheus SQL 导出器 SSM 软件包的先决条件

- 2.3.1550.0 版或更高版本的 SSM Agent 已安装

- 在启用了 SQL Server 用户身份验证的 Windows 上运行 SQL Server 的 Amazon EC2 实例。
- 具有以下权限的 SQL Server 用户：

```
GRANT VIEW ANY DEFINITION TO
```

```
GRANT VIEW SERVER STATE TO
```

- 包含使用 AWS Secrets Manager 的数据库连接字符串的秘密。有关如何创建密钥的更多信息，请参阅 AWS Secrets Manager 用户指南中的创建密钥。密钥必须采用以下格式：

```
{
  "data_source_name":"sqlserver://<username>:<password>@localhost:1433"
}
```

> **Note**
>
> 如果密码或用户名包含特殊字符，则必须对特殊字符进行百分号编码，以确保成功连接到数据库。

安装和配置 **AWSObservabilityExporter-SQLExporterInstallAndConfigure** 软件包

AWSObservabilityExporter-SQLExporterInstallAndConfigure 软件包是一个 SSM Distributor 软件包，您可以使用它来安装和配置 SQL Prometheus 指标导出器。当 Prometheus 导出器发送 Java 指标时，可以配置 CloudWatch 代理来检索 CloudWatch 服务的指标。

1. 根据您的偏好，准备 SQL 导出器 YAML 配置。以下示例配置配置了一项指标。使用示例配置借助其他指标更新配置或创建自己的配置。

```
---
global:
  scrape_timeout_offset: 500ms
  min_interval: 0s
  max_connections: 3
  max_idle_connections: 3
target:
  aws_secret_name: <SECRET_NAME>
  collectors:
    - mssql_standard
```

```
collectors:
  - collector_name: mssql_standard
    metrics:
    - metric_name: mssql_batch_requests
      type: counter
      help: 'Number of command batches received.'
      values: [cntr_value]
      query: |
        SELECT cntr_value
        FROM sys.dm_os_performance_counters WITH (NOLOCK)
        WHERE counter_name = 'Batch Requests/sec'
```

2. 将编码为 Base64 的 Prometheus SQL 导出器 YAML 配置文件复制到 SSM Parameter Store 中的新 SSM 参数。
3. 导航到 SSM Distributor 控制台，然后打开 Owned by Amazon（Amazon 所拥有）选项卡。选择 AWSObservabilityExporter-SQLExporterInstallAndConfigure，然后选择一次性安装。
4. 将“其他参数”替换为以下信息。SSM_PARAMETER_NAME 是您在步骤 2 中创建的参数的名称。

```
{
  "SSM_EXPORTER_CONFIGURATION":
    "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
    "SSM_PROMETHEUS_PORT": "9399",
    "SSM_WORKLOAD_NAME": "SQL"
}
```

5. 选择包含 SQL Server 数据库的 Amazon EC2 实例，然后选择运行。

CloudWatch Application Insights 使用的 AWS Systems Manager（SSM）文档

Application Insights 使用本节中列出的 SSM 文档来定义 AWS Systems Manager 对托管实例执行的操作。这些文档使用 Systems Manager 的 Run Command 功能来自动完成执行 Application Insights 监控功能所需的任务。这些文档的运行时间表由 Application Insights 维护，无法更改。

有关 SSM 文档的更多信息，请参阅《AWS Systems Manager 用户指南》中的 AWS Systems Manager 文档。

由 CloudWatch Application Insights 管理的文档

下表列出了由 Application Insights 管理的 SSM 文档。

文档名称	描述	运行时间表
AWSEC2-DetectWorkload	自动检测在您的应用程序环境中运行的应用程序，这些应用程序可以设置为由 Application Insights 监控。	本文档每小时在您的应用程序环境中运行一次，以获取最新的应用程序详细信息。
AWSEC2-CheckPerformanceCounterSets	检查您的 Amazon EC2 Windows 实例上是否启用了性能计数器命名空间。	本文档在您的应用程序环境中每小时运行一次，并且仅在启用了相应的命名空间时才监控性能计数器指标。
AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure	根据您的应用程序组件的监控配置来安装和配置 CloudWatch 代理。	本文档每 30 分钟运行一次，以确保 CloudWatch 代理配置始终是准确和最新的。在更改应用程序监控设置（例如添加或移除指标或更新日志配置）后，该文档也会立即运行。

由 AWS Systems Manager 管理的文档

以下文档由 CloudWatch Application Insights 使用，并由 Systems Manager 管理。

AWS-ConfigureAWSPackage

Application Insights 使用此文档来安装和卸载 Prometheus 导出程序分发服务器软件包，收集特定于工作负载的指标，并全面监控客户 Amazon EC2 实例上的工作负载。只有在您的实例上运行相关的目标工作负载时，CloudWatch Application Insights 才会安装 Prometheus 导出程序分发服务器软件包。

下表列出了 Prometheus 导出程序分发服务器软件包和相关的目标工作负载。

Prometheus 导出程序分发服务器软件包名称	目标工作负载
AWSObservabilityExporter-HAClusterExporterInstallAndConfigure	SAP HANA HA

Prometheus 导出程序分发服务器软件包名称	目标工作负载
AWSObservabilityExporter-JMXExporterInstallAndConfigure	Java/JMX
AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure	SAP HANA
AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure	NetWeaver
AWSObservabilityExporter-SQLExporterInstallAndConfigure	SQL Server（Windows）和 SAP ASE（Linux）

AmazonCloudWatch-ManageAgent

Application Insights 使用此文档来管理您的实例上的 CloudWatch 代理的状态和配置，并跨操作系统从 Amazon EC2 实例收集内部系统级指标和日志。

开始使用 Amazon CloudWatch Application Insights

要开始使用 CloudWatch Application Insights，请确认您满足下面列出的先决条件并创建了 IAM policy。然后，您可以开始使用控制台链接以启用 CloudWatch Application Insights。要配置应用程序资源，请执行设置、配置和管理您的应用程序以进行监控中的步骤。

内容

- 访问 CloudWatch Application Insights
- 先决条件
- IAM policy
- 基于账户的应用程序载入的 IAM 角色权限
- 设置、配置和管理您的应用程序以进行监控

访问 CloudWatch Application Insights

您可以通过以下界面之一访问及管理 CloudWatch Application Insights：

- CloudWatch 控制台。要为应用程序添加监控，请选择 CloudWatch 控制台左侧导航窗格中 Insights 下的 Application Insights。在配置应用程序后，您可以使用 CloudWatch 控制台查看和分析检测到的问题。
- AWS Command Line Interface (AWS CLI)。您可以使用 AWS CLI 访问 AWS API 操作。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的安装 AWS Command Line Interface。有关 Application Insights API 的信息，请参阅 Amazon CloudWatch Application Insights API 参考。

先决条件

您必须完成以下必需任务才能使用 CloudWatch Application Insights 配置应用程序：

- AWS Systems Manager 启用 – 在 Amazon EC2 实例上安装 Systems Manager Agent（SSM Agent），并启用 SSM 实例。有关如何安装 SSM Agent 的信息，请参阅《AWS Systems Manager 用户指南》中的设置 AWS Systems Manager。
- EC2 实例角色 – 您必须附加以下 Amazon EC2 实例角色，才能启用 Systems Manager
  - 要启用 Systems Manager，您必须附加 `AmazonSSMManagedInstanceCore` 角色。有关更多信息，请参阅 AWS Systems Manager 基于身份的策略示例。
  - 要使实例指标和日志通过 CloudWatch 发出，您必须附加 `CloudWatchAgentServerPolicy` 策略。有关更多信息，请参阅创建用于 CloudWatch 代理的 IAM 角色和用户
- AWS Resource Groups – 要将应用程序添加到 CloudWatch Application Insights，请创建一个包含应用程序堆栈所用的所有相关 AWS 资源的资源组。这包括应用程序负载均衡器、运行 IIS 和 Web 前端的 Amazon EC2 实例、.NET 工作线程层和 SQL Server 数据库。有关 Application Insights 支持的应用程序组件和技术堆栈的更多信息，请参阅 支持的应用程序组件。CloudWatch Application Insights 自动包含使用与资源组相同的标签或 CloudFormation 堆栈的自动扩缩组，因为 CloudFormation 资源组不支持自动扩缩组。有关更多信息，请参阅 AWS Resource Groups 入门。
- IAM 权限 – 对于没有管理访问权限的用户，必须创建允许 Application Insights 创建服务相关角色的 AWS Identity and Access Management（IAM）policy，并将其附加到用户身份。有关如何创建 IAM policy 的更多信息，请参阅 IAM policy。
- 服务相关角色 – Application Insights 使用 AWS Identity and Access Management（IAM）服务相关角色。当您在 Application Insights 控制台中创建首个 Application Insights 应用程序时，将会为您创建服务相关角色。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。
- EC2 Windows 实例对性能计数器指标的支持 – 要在 Amazon EC2 Windows 实例上监控性能计数器指标，必须在实例上安装性能计数器。有关性能计数器指标和相应的性能计数器集名称，请参阅性能计数器指标。有关性能计数器的更多信息，请参阅性能计数器。

- Amazon CloudWatch 代理 – Application Insights 安装和配置 CloudWatch 代理。如果您安装了 CloudWatch 代理，则 Application Insights 会保留您的配置。为避免合并冲突，请从现有 CloudWatch 代理配置文件中删除您想在 Application Insights 中使用的资源的配置。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

IAM policy

要使用 CloudWatch Application Insights，您必须创建一个 AWS Identity and Access Management（IAM）policy，并将其附加到您的用户、组或角色。有关用户、组和角色的更多信息，请参阅 IAM 身份（用户、用户组和角色）。IAM policy 定义了用户权限。

使用控制台创建 IAM policy

要使用 IAM 控制台创建 IAM policy，请执行以下步骤。

1. 转到 IAM 控制台。在左侧导航窗格中，选择策略。
2. 在页面顶部，选择创建策略。
3. 选择 JSON 选项卡。
4. 在 JSON 选项卡下面复制并粘贴以下 JSON 文档。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "applicationinsights:*",
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "resource-groups:ListGroups"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

5. 选择查看策略。
6. 输入策略的名称，例如“AppInsightsPolicy”。（可选）输入描述。
7. 选择 Create Policy (创建策略)。

8. 在左侧的导航窗格中，选择用户组、用户或角色。
9. 选择要附加策略的用户组、用户或角色的名称。
10. 选择添加权限。
11. 选择 Attach existing policies directly（直接附加现有策略）。
12. 搜索刚创建的策略，然后选中策略名称左侧的复选框。
13. 选择下一步: 审核。
14. 确保列出了正确的策略，然后选择添加权限。
15. 在使用 CloudWatch Application Insights 时，请确保使用与刚创建的策略关联的用户登录。

使用 AWS CLI 创建 IAM policy

要使用 AWS CLI 创建 IAM policy，请使用上面的 JSON 文档作为当前文件夹中的文件，从命令行运行 create-policy 操作。

使用 AWS Tools for Windows PowerShell 创建 IAM policy

要使用 AWS Tools for Windows PowerShell 创建 IAM policy，请使用上面的 JSON 文档作为当前文件夹中的文件，运行 New-IAMPolicy cmdlt。

基于账户的应用程序载入的 IAM 角色权限

如果您想载入账户中的所有资源，并选择不使用 Application Insights 托管式策略来完全访问 Application Insights 功能，则必须向 IAM 角色附加以下权限，以便 Application Insights 可以发现您账户中的所有资源：

```
"ec2:DescribeInstances"
"ec2:DescribeNatGateways"
"ec2:DescribeVolumes"
"ec2:DescribeVPCs"
"rds:DescribeDBInstances"
"rds:DescribeDBClusters"
"sqs:ListQueues"
"elasticloadbalancing:DescribeLoadBalancers"
"autoscaling:DescribeAutoScalingGroups"
"lambda:ListFunctions"
"dynamodb:ListTables"
"s3:ListAllMyBuckets"
"sns:ListTopics"
```

```
"states:ListStateMachines"
"apigateway:GET"
"ecs:ListClusters"
"ecs:DescribeTaskDefinition"
"ecs:ListServices"
"ecs:ListTasks"
"eks:ListClusters"
"eks:ListNodegroups"
"fsx:DescribeFileSystems"
"route53:ListHealthChecks"
"route53:ListHostedZones"
"route53:ListQueryLoggingConfigs"
"route53resolver:ListFirewallRuleGroups"
"route53resolver:ListFirewallRuleGroupAssociations"
"route53resolver:ListResolverEndpoints"
"route53resolver:ListResolverQueryLogConfigs"
"route53resolver:ListResolverQueryLogConfigAssociations"
"logs:DescribeLogGroups"
"resource-explorer:ListResources"
```

设置、配置和管理您的应用程序以进行监控

本节提供使用控制台、AWS CLI 和 AWS Tools for Windows PowerShell 来设置、配置和管理 CloudWatch Application Insights 应用程序的步骤。

主题

- 设置、配置和管理您的应用程序以从 CloudWatch 控制台进行监控
- 使用命令行设置、配置和管理您的应用程序以进行监控
- 检测到的问题的 Application CloudWatch Events 和通知

设置、配置和管理您的应用程序以从 CloudWatch 控制台进行监控

本节提供设置、配置和管理应用程序以从 CloudWatch 控制台进行监控的步骤。

控制台程序

- 添加和配置应用程序
- 为 Amazon ECS 和 Amazon EKS 资源监控启用 Application Insights
- 为应用程序组件禁用监控
- 删除应用程序

添加和配置应用程序

从 CloudWatch 控制台添加和配置应用程序

要通过 CloudWatch 控制台开始使用 CloudWatch Application Insights，请执行以下步骤。

1. 开始。打开 CloudWatch 控制台登陆页面。从左侧导航窗格中，选择 Insights 下的 Application Insights。所打开的页面将显示使用 CloudWatch Application Insights 监控的应用程序列表及它们的监控状态。
2. 添加应用程序。要为应用程序设置监控，请选择 Add an application（添加应用程序）。在选择 Add an application（添加应用程序）时，系统会提示您 Choose Application Type（选择应用程序类型）。

   - 基于资源组的应用程序。选择此选项后，您可以选择此账户中要监控的资源组。要在一个组件上使用多个应用程序，您必须使用基于资源组的监控。
   - 基于账户的应用程序。选择此选项后，您可以监控此账户中的所有资源。如果您想监控账户中的所有资源，我们建议使用此选项而不是基于资源组的选项，因为应用程序载入过程更快。

   > **Note**
   >
   > 不能使用 Application Insights 将基于资源组的监控与基于账户的监控相结合。要更改应用程序类型，必须删除所有正在监控之下的应用程序，并 Choose Application Type（选择应用程序类型）。

   当您添加第一个应用程序以进行监控时，CloudWatch Application Insights 会在账户中创建服务相关角色，该角色授予 Application Insights 代表您调用其他 AWS 服务的权限。有关 Application Insights 在您的账户中创建的服务相关角色的更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。
3. Resource-based application monitoring

   1. 选择资源组。在 Specify application details（指定应用程序详细信息）页面上，从下拉列表中选择包含您应用程序资源的 AWS 资源组。这些资源包括前端服务器、负载均衡器、Auto Scaling 组和数据库服务器。

      如果还没有为应用程序创建资源组，则可以选择 Create new resource group（创建新资源组）来创建一个资源组。有关创建资源组的详细信息，请参阅 AWS Resource Groups 用户指南。

2. 监控 CloudWatch Events。选中复选框以将 Application Insights 监控与 CloudWatch Events 集成，获取 Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health API 和通知、Amazon RDS、Amazon S3 和 AWS Step Functions 中的洞察。
3. 与 AWS Systems Manager OpsCenter 集成。要在检测到所选应用程序的问题时查看问题并获得通知，请选中 Generate Systems Manager OpsCenter OpsItems for remedial actions（生成 Systems Manager OpsCenter OpsItems 以采取修复措施）复选框。要跟踪用于解析与 AWS 资源相关的操作工作项 (OpsItem) 的操作，请提供 SNS 主题 ARN。
4. 标签（可选）。CloudWatch Application Insights 支持基于标签和基于 CloudFormation 的资源组（Auto Scaling 组除外）。有关更多信息，请参阅使用标签编辑器。
5. 选择下一步。

   将按以下格式为应用程序生成 ARN：

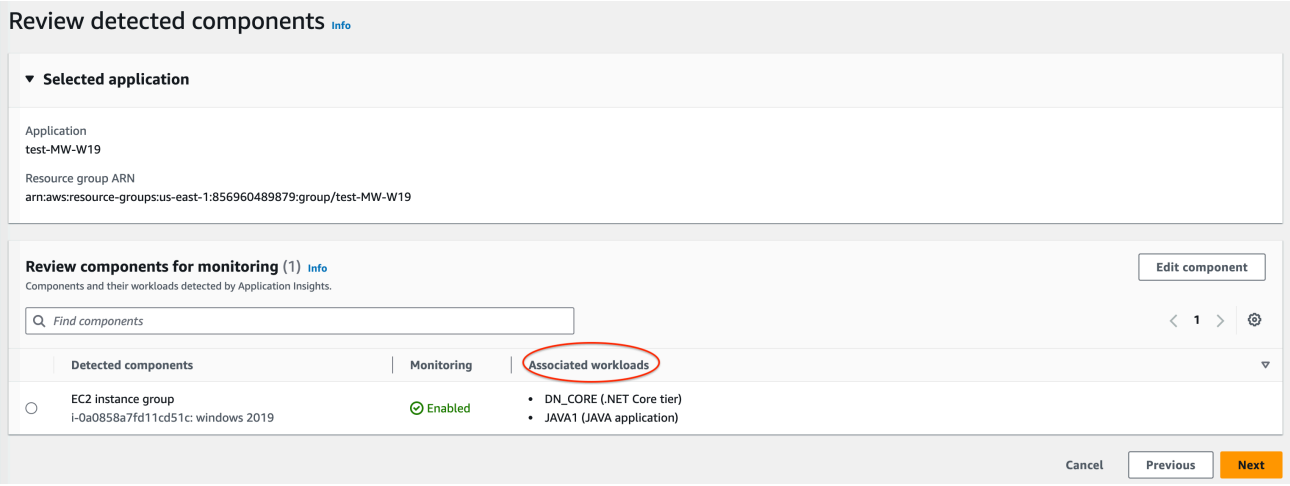
   ```
   arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
   ```

   示例

   ```
   arn:aws:applicationinsights:us-east-1:123456789012:application/resource-group/my-resource-group
   ```

6. 在查看检测到的组件页面的查看要监控的组件下，表列出了检测到的组件及其关联的检测到的工作负载。

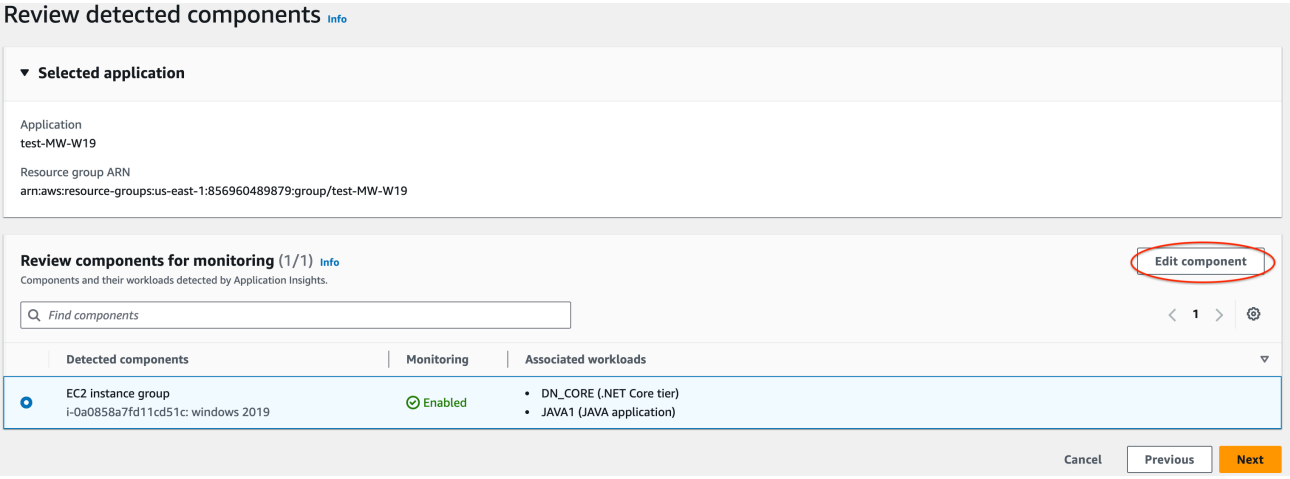
   > **Note**
   >
   > 对于支持多个自定义工作负载的组件，每个组件最多可以监控五个工作负载。这些工作负载将与组件分开监控。



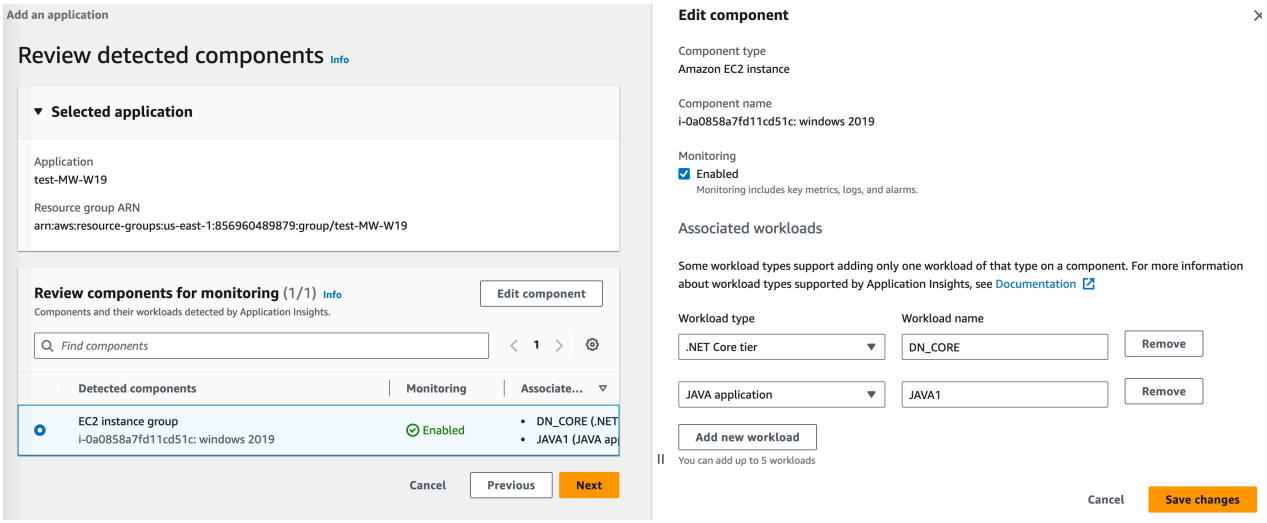
在关联的工作负载下，如果未列出工作负载，则会显示几条可能的消息。

- 无法检测工作负载 – 尝试检测工作负载时出现问题。确保您已完成 先决条件。如果需要添加工作负载，请选择编辑组件。
- 未检测到工作负载 – 我们未检测到任何工作负载。您可能需要添加工作负载。为此，请选择编辑组件。
- 不适用 — 组件不支持自定义工作负载，将使用默认指标、警报和日志进行监控。您不能向这些组件添加工作负载。

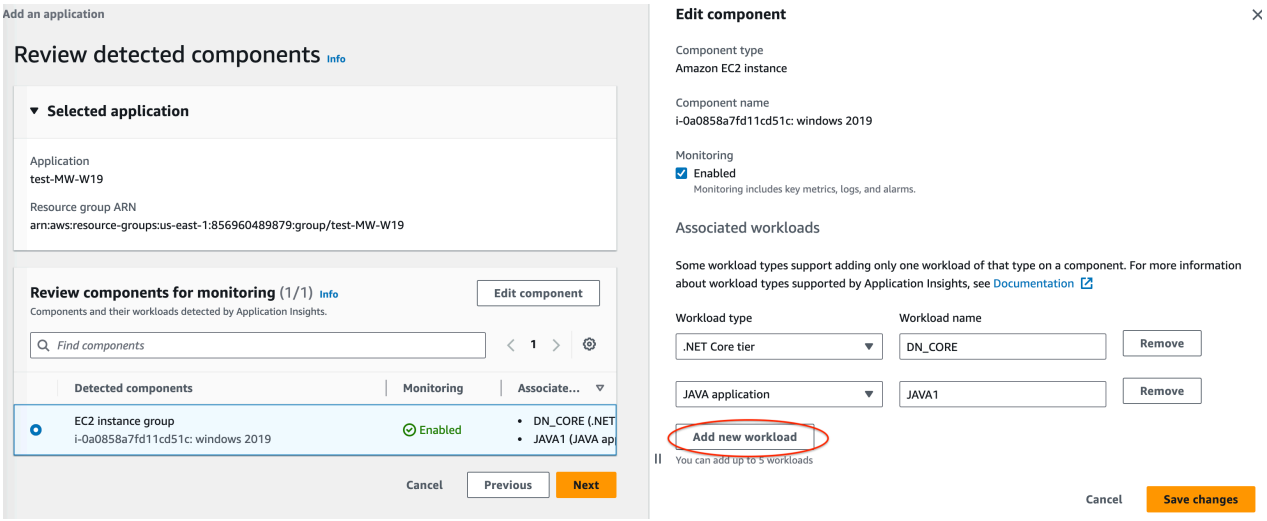
7. 要编辑组件，请选择一个组件，然后选择编辑组件。侧面板打开，显示在组件上检测到的工作负载。在此面板中，您可以编辑组件详细信息并添加新的工作负载。



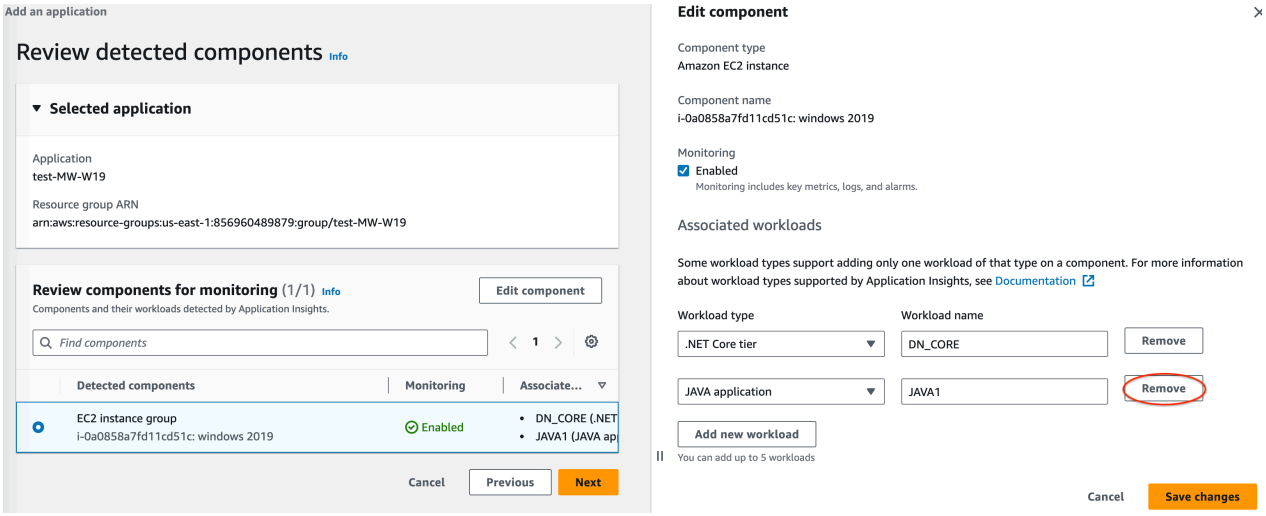
- 如需编辑工作负载类型或名称，请使用下拉列表。



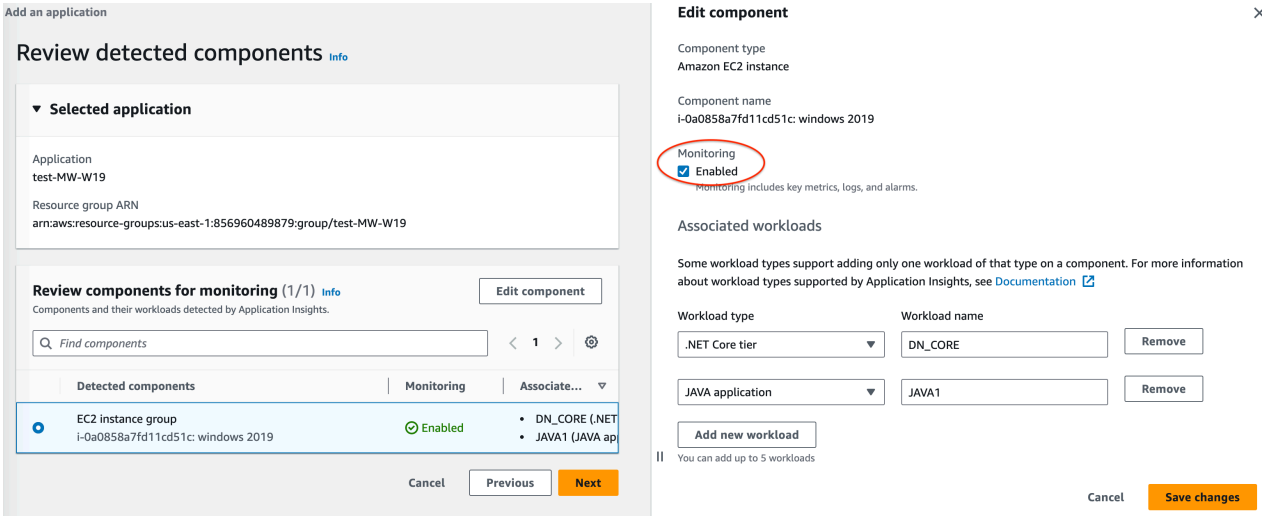
- 要向组件添加工作负载，请选择添加新的工作负载。



  - 如果未出现添加新工作负载，则此组件不支持多个工作负载。
  - 如果未出现关联工作负载标题，则此组件不支持自定义的工作负载。
- 要删除工作负载，请选择要从监控中删除的工作负载旁边的删除。



- 要禁用对整个组件的监控，请清除监控复选框。



- 编辑完组件后，选择右下角的保存更改。组件工作负载的任何更改都将显示在关联的工作负载下的查看要监控的组件表中。

8. 在查看检测到的组件页面上，选择下一步。
9. 指定组件详细信息页面包含上一步中具有可自定义关联工作负载的所有组件。

> **Note**
>
> 如果组件标题具有可选标签，则该组件中工作负载的其他详细信息是可选的。

如果某个组件未出现在此页面上，则该组件没有任何可在此步骤中指定的其他详细信息。

10选择下一步。

11.在查看并提交页面上，查看所有受监控的组件和工作负载详细信息。

12选择提交。

Account-based application monitoring

1. 应用程序名称。为基于账户的应用程序输入名称。
2. 自动监控新资源。默认情况下，Application Insights 使用推荐的设置来配置对载入应用程序后添加到账户的资源组件的监控。通过清除复选框，可以排除对载入应用程序后添加的资源的监控。
3. 监控 CloudWatch Events。选中复选框以将 Application Insights 监控与 CloudWatch Events 集成，获取 Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health API 和通知、Amazon RDS、Amazon S3 和 AWS Step Functions 中的洞察。
4. 与 AWS Systems Manager OpsCenter 集成。要在检测到所选应用程序的问题时查看问题并获得通知，请选中 Generate Systems Manager OpsCenter OpsItems for remedial actions（生成 Systems Manager OpsCenter OpsItems 以采取修复措施）复选框。要跟踪用于解析与 AWS 资源相关的操作工作项 (OpsItem) 的操作，请提供 SNS 主题 ARN。
5. 标签（可选）。CloudWatch Application Insights 支持基于标签和基于 CloudFormation 的资源组（Auto Scaling 组除外）。有关更多信息，请参阅使用标签编辑器。
6. 发现的资源。在您的账户中发现的所有资源都会添加到此列表中。如果 Application Insights 无法发现您账户中的所有资源，则会在页面顶部显示一条错误消息。此消息中包含一个指向有关如何添加所需权限的文档的链接。
7. 选择下一步。

   将按以下格式为应用程序生成 ARN：

   ```
   arn:partition:applicationinsights:region:account-id:application/
   TBD/application-name
   ```

   示例

   ```
   arn:aws:applicationinsights:us-east-1:123456789012:application/TBD/my-
   application
   ```

4. 提交应用程序监控配置后，您将转到应用程序的详细信息页面，您可以在该页面查看 Application summary（应用程序摘要）、Monitored components（已监控组件）的列表和 Unmonitored components（未监控组件）的列表，并通过选择 Components（组件）旁边的选项卡，查看

Configuration history（配置历史记录）、Log patterns（日志模式）和您已应用的任何 Tags（标签）。

要查看关于应用程序的洞察，请选择 View Insights（查看洞察）。

通过选择 Edit（编辑），您可以更新 CloudWatch Events 监控的选项以及与 AWS Systems Manager OpsCenter 的集成。

在 Components（组件）下，您可以选择 Actions（操作）菜单来创建、修改或取消分组实例组。

您可以通过选择组件旁边的项目符号并选择 Manage monitoring（管理监控）来管理组件的监控，包括应用程序层、日志组、事件日志、指标和自定义告警。

为 Amazon ECS 和 Amazon EKS 资源监控启用 Application Insights

您可以启用 Application Insights，以从 Container Insights 控制台监控容器化应用程序和微服务。Application Insights 支持监控以下资源：

- Amazon ECS 集群
- Amazon ECS 服务
- Amazon ECS 任务
- Amazon EKS 集群

启用 Application Insights 后，它会提供建议的指标和日志，检测潜在问题，生成 CloudWatch Events，以及为容器化应用程序和微服务创建自动控制面板。

您可以从 Container Insights 或 Application Insights 控制台为容器化资源启用 Application Insights。

从 Container Insights 控制台启用 Application Insights

在 Container Insights 控制台中的 Container Insights Performance monitoring（性能监控）控制面板上，选择 Auto-configure Application Insights（自动配置 Application Insights）。启用 Application Insights 后，它会显示有关检测到的问题的详细信息。

从 Application Insights 控制台启用 Application Insights

当组件列表中出现 ECS 集群时，Application Insights 会自动使用 Container Insights 启用其他容器监控。

对于 EKS 集群，您可以通过 Container Insights 启用其他监控，提供诊断信息（如容器重启失败），以帮助您查明问题并解决问题。要为 EKS 设置 Container Insights，还需要执行其他步骤。有关详细信息，请参阅 在 Amazon EKS 和 Kubernetes 上设置 Container Insights 了解在 EKS 上设置 Container Insights 的步骤。

使用 EKS 的 Linux 实例支持使用 Container Insights 对 EKS 进行额外监控。

有关对 ECS 和 EKS 集群的 Container Insights 支持的更多信息，请参阅 Container Insights。

为应用程序组件禁用监控

要为应用程序组件禁用监控，请从应用程序详细信息页面中选择要禁用监控的组件。选择 Actions（操作），然后 Remove from monitoring（取消监控）。

删除应用程序

要删除应用程序，请在 CloudWatch 控制面板中的左侧导航窗格中选择 Insights 下方的 Application Insights。选择要删除的应用程序。在 Actions（操作）下选择 Delete application（删除应用程序）。这会删除监控，并删除为应用程序组件保存的所有监视器。不会删除应用程序资源。

使用命令行设置、配置和管理您的应用程序以进行监控

本节提供设置、配置和管理应用程序以使用 AWS CLI 和 AWS Tools for Windows PowerShell 进行监控的步骤。

命令行程序

- 添加和管理应用程序
- 管理和更新监控
- 针对 SQL Always On 可用性组配置监控
- 为 MySQL RDS 配置监控
- 为 MySQL EC2 配置监控
- 为 PostgreSQL RDS 配置监控
- 为 PostgreSQL EC2 配置监控
- 为 Oracle RDS 配置监控
- 为 Oracle EC2 配置监控

添加和管理应用程序

您可以使用命令行添加、管理和配置 Application Insights 应用程序以及获取有关它的信息。

主题

- 添加应用程序
- 描述应用程序
- 列出应用程序中的组件
- 描述组件
- 将类似资源分组到自定义组件中
- 对自定义组件取消分组
- 更新应用程序
- 更新自定义组件

添加应用程序

使用 AWS CLI 添加应用程序

要使用 AWS CLI 为名为 `my-resource-group` 的资源组添加应用程序，并启用 OpsCenter 以将创建的 opsItem 传递到 SNS 主题 ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`，请使用以下命令。

```
aws application-insights create-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

使用 AWS Tools for Windows PowerShell 添加应用程序

要使用 AWS Tools for Windows PowerShell 为名为 `my-resource-group` 的资源组添加应用程序，并启用 OpsCenter 以将创建的 opsItem 传递到 SNS 主题 ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`，请使用以下命令。

```
New-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

描述应用程序

使用 AWS CLI 描述应用程序

要使用 AWS CLI 描述在名为 `my-resource-group` 的资源组上创建的应用程序，请使用以下命令。

```
aws application-insights describe-application --resource-group-name my-resource-group
```

使用 AWS Tools for Windows PowerShell 描述应用程序

要使用 AWS Tools for Windows PowerShell 描述在名为 my-resource-group 的资源组上创建的应用程序，请使用以下命令。

```
Get-CWAIApplication -ResourceGroupName my-resource-group
```

列出应用程序中的组件

使用 AWS CLI 列出应用程序中的组件

要使用 AWS CLI 列出在名为 my-resource-group 的资源组上创建的组件，请使用以下命令。

```
aws application-insights list-components --resource-group-name my-resource-group
```

使用 AWS Tools for Windows PowerShell 列出应用程序中的组件

要使用 AWS Tools for Windows PowerShell 列出在名为 my-resource-group 的资源组上创建的组件，请使用以下命令。

```
Get-CWAIComponentList -ResourceGroupName my-resource-group
```

描述组件

使用 AWS CLI 描述组件

您可以使用以下 AWS CLI 命令描述属于在名为 my-resource-group 的资源组上创建的应用程序且名为 my-component 的组件。

```
aws application-insights describe-component --resource-group-name my-resource-group --
component-name my-component
```

使用 AWS Tools for Windows PowerShell 描述组件

您可以使用以下 AWS Tools for Windows PowerShell 命令描述属于在名为 my-resource-group 的资源组上创建的应用程序且名为 my-component 的组件。

```
Get-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

将类似资源分组到自定义组件中

我们建议将类似的资源（如 .NET Web 服务器实例）分组到自定义组件中，以轻松添加资源以及更好地进行监控和提供相应的信息。目前，CloudWatch Application Insights 支持 EC2 实例的自定义组。

使用 AWS CLI 将资源分组到自定义组件

要为名为 my-resource-group 的资源组创建的应用程序使用 AWS CLI 将三个实例（arn:aws:ec2:us-east-1:123456789012:instance/i-11111、arn:aws:ec2:us-east-1:123456789012:instance/i-22222 和 arn:aws:ec2:us-east-1:123456789012:instance/i-33333）一起分组到一个名为 my-component 的自定义组件，请使用以下命令。

```
aws application-insights create-component --resource-group-name my-
resource-group --component-name my-component --resource-list arn:aws:ec2:us-
east-1:123456789012:instance/i-11111 arn:aws:ec2:us-east-1:123456789012:instance/
i-22222 arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

使用 AWS Tools for Windows PowerShell 将资源分组到自定义组件

要为名为 my-resource-group 的资源组创建的应用程序使用 AWS Tools for Windows PowerShell 将三个实例（arn:aws:ec2:us-east-1:123456789012:instance/i-11111、arn:aws:ec2:us-east-1:123456789012:instance/i-22222 和 arn:aws:ec2:us-east-1:123456789012:instance/i-33333）一起分组到一个名为 my-component 的自定义组件，请使用以下命令。

```
New-CWAIComponent -ResourceGroupName my-resource-group -ComponentName my-component
 -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-11111,arn:aws:ec2:us-
east-1:123456789012:instance/i-22222,arn:aws:ec2:us-east-1:123456789012:instance/
i-33333
```

对自定义组件取消分组

使用 AWS CLI 对自定义组件取消分组

要使用 AWS CLI 对在资源组 my-resource-group 上创建的应用程序中名为 my-component 的自定义组件取消分组，请使用以下命令。

```
aws application-insights delete-component --resource-group-name my-resource-group --
component-name my-new-component
```

使用 AWS Tools for Windows PowerShell 对自定义组件取消分组

要使用 AWS Tools for Windows PowerShell 对在资源组 `my-resource-group` 上创建的应用程序中名为 `my-component` 的自定义组件取消分组，请使用以下命令。

```
Remove-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

更新应用程序

使用 AWS CLI 更新应用程序

您可以使用 AWS CLI 更新应用程序，以针对在应用程序中检测到的问题生成 AWS Systems Manager OpsCenter OpsItem，并使用以下命令将创建的 OpsItem 与 SNS 主题 `arn:aws:sns:us-east-1:123456789012:MyTopic` 相关联。

```
aws application-insights update-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

使用 AWS Tools for Windows PowerShell 更新应用程序

您可以使用 AWS Tools for Windows PowerShell 更新应用程序，以针对在应用程序中检测到的问题生成 AWS SSM OpsCenter OpsItem，并使用以下命令将创建的 OpsItem 与 SNS 主题 `arn:aws:sns:us-east-1:123456789012:MyTopic` 相关联。

```
Update-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

更新自定义组件

使用 AWS CLI 更新自定义组件

您可以使用 AWS CLI，通过使用以下命令，使用新的组件名称 `my-new-component` 和更新的实例组来更新名为 `my-component` 的自定义组件。

```
aws application-insights update-component --resource-group-name my-resource-
group --component-name my-component --new-component-name my-new-component --
resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-44444 arn:aws:ec2:us-
east-1:123456789012:instance/i-55555
```

使用 AWS Tools for Windows PowerShell 更新自定义组件

您可以使用 AWS Tools for Windows PowerShell，通过使用以下命令，使用新的组件名称 my-new-component 和更新的实例组来更新名为 my-component 的自定义组件。

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-
component -ResourceGroupName my-resource-group -ResourceList arn:aws:ec2:us-
east-1:123456789012:instance/i-44444,arn:aws:ec2:us-east-1:123456789012:instance/
i-55555
```

管理和更新监控

您可以使用命令行管理和更新针对 Application Insights 应用程序的监控。

主题

- 列出应用程序的问题
- 描述应用程序问题
- 描述与问题关联的异常或错误
- 描述应用程序的异常或错误
- 描述组件的监控配置
- 描述建议的组件监控配置
- 更新组件的监控配置
- 对指定的资源组取消 Application Insights 监控

列出应用程序的问题

使用 AWS CLI 列出应用程序的问题

要使用 AWS CLI 列出自 Unix Epoch 以来针对在名为 my-resource-group 的资源组上创建的应用程序在 1000 到 10000 毫秒之间检测到的应用程序问题，请使用以下命令。

```
aws application-insights list-problems --resource-group-name my-resource-group --start-
time 1000 --end-time 10000
```

使用 AWS Tools for Windows PowerShell 列出应用程序的问题

要使用 AWS Tools for Windows PowerShell 列出自 Unix Epoch 以来针对在名为 my-resource-group 的资源组上创建的应用程序在 1000 到 10000 毫秒之间检测到的应用程序问题，请使用以下命令。

```
$startDate = "8/6/2019 3:33:00"
$endDate = "8/6/2019 3:34:00"
Get-CWAIProblemList -ResourceGroupName my-resource-group -StartTime $startDate -
EndTime $endDate
```

描述应用程序问题

使用 AWS CLI 描述应用程序问题

要使用 AWS CLI 描述问题 ID 为 p-1234567890 的问题，请使用以下命令。

```
aws application-insights describe-problem –problem-id p-1234567890
```

使用 AWS Tools for Windows PowerShell 描述应用程序问题

要使用 AWS Tools for Windows PowerShell 描述问题 ID 为 p-1234567890 的问题，请使用以下命令。

```
Get-CWAIProblem -ProblemId p-1234567890
```

描述与问题关联的异常或错误

使用 AWS CLI 描述与问题关联的异常或错误

要使用 AWS CLI 描述与问题 ID 为 p-1234567890 的问题关联的异常或错误，请使用以下命令。

```
aws application-insights describe-problem-observations --problem-id -1234567890
```

使用 AWS Tools for Windows PowerShell 描述与问题关联的异常或错误

要使用 AWS Tools for Windows PowerShell 描述与问题 ID 为 p-1234567890 的问题关联的异常或错误，请使用以下命令。

```
Get-CWAIProblemObservation -ProblemId p-1234567890
```

描述应用程序的异常或错误

使用 AWS CLI 描述应用程序的异常或错误

要使用 AWS CLI 描述观察 ID 为 o-1234567890 的应用程序的异常或错误，请使用以下命令。

```
aws application-insights describe-observation –observation-id o-1234567890
```

使用 AWS Tools for Windows PowerShell描述应用程序的异常或错误

要使用 AWS Tools for Windows PowerShell 描述观察 ID 为 o-1234567890 的应用程序的异常或错误，请使用以下命令。

```
Get-CWAIObservation -ObservationId o-1234567890
```

描述组件的监控配置

使用 AWS CLI 组件描述组件的监控配置

要使用 AWS CLI 描述在资源组 my-resource-group 上创建的应用程序中名为 my-component 的组件的监控配置，请使用以下命令。

```
aws application-insights describe-component-configuration –resource-group-name my-
resource-group –component-name my-component
```

使用 AWS Tools for Windows PowerShell 描述组件的监控配置

要使用 AWS Tools for Windows PowerShell 描述在资源组 my-resource-group 上创建的应用程序中名为 my-component 的组件的监控配置，请使用以下命令。

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group
```

有关组件配置和示例 JSON 文件的更多信息，请参阅使用组件配置。

描述建议的组件监控配置

使用 AWS CLI 描述建议的组件监控配置

当组件是 .NET Worker 应用程序的一部分时，您可以使用 AWS CLI，通过使用以下命令来描述在资源组 my-resource-group 上创建的应用程序中名为 my-component 的组件的建议监控配置。

```
aws application-insights describe-component-configuration-recommendation --resource-
group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

使用 AWS Tools for Windows PowerShell 描述建议的组件监控配置

当组件是 .NET Worker 应用程序的一部分时，您可以使用 AWS Tools for Windows PowerShell，通过使用以下命令来描述在资源组 my-resource-group 上创建的应用程序中名为 my-component 的组件的建议监控配置。

```
Get-CWAIComponentConfigurationRecommendation -ComponentName my-component -
ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

有关组件配置和示例 JSON 文件的更多信息，请参阅使用组件配置。

更新组件的监控配置

使用 AWS CLI 更新组件的监控配置

要使用 AWS CLI 更新在名为 my-resource-group 的资源组上创建的应用程序中名为 my-component 的组件，请使用以下命令。该命令包括以下操作：

1. 启用组件监控。
2. 将组件层设置为 .NET Worker。
3. 更新组件的 JSON 配置以从本地文件 configuration.txt 中读取。

```
aws application-insights update-component-configuration --resource-group-name my-
resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --
component-configuration "file://configuration.txt"
```

使用 AWS Tools for Windows PowerShell 更新组件的监控配置

要使用 AWS Tools for Windows PowerShell 更新在名为 my-resource-group 的资源组上创建的应用程序中名为 my-component 的组件，请使用以下命令。该命令包括以下操作：

1. 启用组件监控。
2. 将组件层设置为 .NET Worker。
3. 更新组件的 JSON 配置以从本地文件 configuration.txt 中读取。

```
[string]$config = Get-Content -Path configuration.txt
Update-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

有关组件配置和示例 JSON 文件的更多信息，请参阅使用组件配置。

对指定的资源组取消 Application Insights 监控

使用 AWS CLI 对指定的资源组取消 Application Insights 监控

要使用 AWS CLI 对在名为 `my-resource-group` 的资源组上创建的应用程序取消监控，请使用以下命令。

```
aws application-insights delete-application --resource-group-name my-resource-group
```

使用 AWS Tools for Windows PowerShell 对指定的资源组取消 Application Insights 监控

要使用 AWS Tools for Windows PowerShell 对在名为 `my-resource-group` 的资源组上创建的应用程序取消监控，请使用以下命令。

```
Remove-CWAIApplication -ResourceGroupName my-resource-group
```

针对 SQL Always On 可用性组配置监控

1. 使用 SQL HA EC2 实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 通过创建新的应用程序组件，定义表示 SQL HA 集群的 EC2 实例。

```
aws application-insights create-component --resource-group-name
 "<RESOURCE_GROUP_NAME>" --component-name SQL_HA_CLUSTER --resource-list
  "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>"
 "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>
```

3. 配置 SQL HA 组件。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "SQL_HA_CLUSTER" --
monitor --tier SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP --monitor  --component-
configuration '{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
```

```
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/
sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
```

```
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-
<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\
\MSSQL\\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}'
```

> **Note**
>
> Application Insights 必须获取应用程序事件日志（信息级别）才能检测集群活动（如故障转移）。

为 MySQL RDS 配置监控

1. 使用 RDS MySQL 数据库实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 默认情况下，启用错误日志。可以使用数据参数组启用慢速查询日志。有关更多信息，请参阅访问 MySQL 慢速查询和常规日志。
   - `set slow_query_log = 1`
   - `set log_output = FILE`
3. 将要监控的日志导出到 CloudWatch Logs 中。有关更多信息，请参阅将 MySQL 日志发布到 CloudWatch Logs。
4. 配置 MySQL RDS 组件。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier DEFAULT --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logType\":
\"MYSQL\",\"monitor\":true},{\"logType\": \"MYSQL_SLOW_QUERY\",\"monitor\":false}]}"
```

为 MySQL EC2 配置监控

1. 使用 SQL HA EC2 实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 默认情况下，启用错误日志。可以使用数据参数组启用慢速查询日志。有关更多信息，请参阅访问 MySQL 慢速查询和常规日志。
   - `set slow_query_log = 1`
   - `set log_output = FILE`
3. 配置 MySQL EC2 组件。

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier MYSQL --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logGroupName
```

```
\":\"<UNIQUE_LOG_GROUP_NAME>\",\"logPath\":\"C:\\\\ProgramData\\\\MySQL\\\\MySQL
 Server **\\\\Data\\\\<FILE_NAME>.err\",\"logType\":\"MYSQL\",\"monitor\":true,
\"encoding\":\"utf-8\"}]}"
```

为 PostgreSQL RDS 配置监控

1. 使用 Postgre RDS 数据库实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 默认情况下，没有启用将 PostgreSQL 日志发布到 CloudWatch 的功能。要启用监控，请打开 RDS 控制台并选择要监控的数据库。选择右上角的 Modify（修改），然后选中标注为 PostgreSQL 日志的复选框。选择 Continue（继续）以保存此设置。
3. 您的 PostgreSQL 日志将导出到 CloudWatch。
4. 配置 PostgreSQL RDS 组件。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"POSTGRESQL\",
         \"monitor\": true
      }
   ]
}"
```

为 PostgreSQL EC2 配置监控

1. 使用 PostgreSQL EC2 实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 配置 PostgreSQL EC2 组件。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier POSTGRESQL --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/var/lib/pgsql/data/log/\",
         \"logType\":\"POSTGRESQL\",
         \"monitor\":true,
         \"encoding\":\"utf-8\"
      }
   ]
}"
```

为 Oracle RDS 配置监控

1. 使用 Oracle RDS 数据库实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 默认情况下，没有启用将 Oracle 日志发布到 CloudWatch 的功能。要启用监控，请打开 RDS 控制台并选择要监控的数据库。选择右上角的 Modify（修改），然后选中标注为 Alert（提示）日志和 Listener（侦听器）日志的复选框。选择 Continue（继续）以保存此设置。
3. 您的 Oracle 日志将导出到 CloudWatch。
4. 配置 Oracle RDS 组件。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"ORACLE_ALERT\",
         \"monitor\": true
      },
      {
         \"logType\": \"ORACLE_LISTENER\",
         \"monitor\": true
      }
   ]
}"
```

为 Oracle EC2 配置监控

1. 使用 Oracle EC2 实例为资源组创建应用程序。

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. 配置 Oracle EC2 组件。

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier ORACLE --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
```

```
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/rdbms/*/*/trace\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      },
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/tnslsnr/$HOSTNAME/listener/trace/\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      }
   ]
}"
```

检测到的问题的 Application CloudWatch Events 和通知

对于添加到 CloudWatch Application Insights 的每个应用程序，将尽可能为以下事件发布 CloudWatch Events：

- 问题创建。在 CloudWatch Application Insights 检测到新问题时发出。
  - 详细信息类型：“检测到 Application Insights 问题”
  - 详细信息：
    - `problemId`：检测到的问题 ID。
    - `region`：创建问题的 AWS 区域。
    - `resourceGroupName`：检测到问题的注册应用程序的资源组。
    - `status`：问题的状态。可能的状态和定义如下：
      - `In progress`：发现了一个新问题。这个问题仍在接受观察。
      - `Recovering`：问题正在趋于稳定。当问题处于此状态时，您可以手动解决问题。
      - `Resolved`：问题已解决。关于这个问题没有新的观察结果。
      - `Recurring`：问题在过去 24 小时内得到解决。经过进一步观察，发现问题重新出现。
    - `severity`：问题的严重性。
    - `problemUrl`：问题的控制台 URL。

- 问题更新。使用新观察结果更新问题或更新现有观察结果并随后更新问题时发出；更新包括问题解决方案或处理措施。
  - 详细信息类型："已更新 Application Insights 问题"
  - 详细信息：
    - `problemId`：创建的问题 ID。
    - `region`：创建问题的 AWS 区域。
    - `resourceGroupName`：检测到问题的注册应用程序的资源组。
    - `status`：问题的状态。
    - `severity`：问题的严重性。
    - `problemUrl`：问题的控制台 URL。

如何接收应用程序生成的问题事件的通知

从 CloudWatch 控制台中，选择 Events（事件）下左侧导航窗格中的 Rule（规则）。从 Rules (规则) 页面中，选择 Create rule (创建规则)。选择 Service Name（服务名称）下拉列表中的 Amazon CloudWatch Application Insights，然后选择 Event Type（事件类型）。然后，选择 Add target (添加目标)，并选择目标和参数，例如 SNS topic (SNS 主题) 或 Lambda function (Lambda 函数)。

通过 AWS Systems Manager 执行的操作。CloudWatch Application Insights 提供与 Systems Manager OpsCenter 的内置集成。如果您选择将此集成用于您的应用程序，则会在 OpsCenter 控制台上为该应用程序检测到的每个问题创建一个 OpsItem。在 OpsCenter 控制台中，您可以查看 CloudWatch Application Insights 检测到的问题的摘要信息，并选择一个 Systems Manager Automation 运行手册来采取补救措施，或者进一步识别导致应用程序中出现资源问题的 Windows 进程。

Application Insights 跨账户可观测性

借助 CloudWatch Application Insights 跨账户可观测性，您可以跨越一个区域内的多个 AWS 账户监控应用程序以及排查问题。

您可以使用 Amazon CloudWatch Observability Access Manager 将一个或多个 AWS 账户设置为监控账户。通过在监控账户中创建接收器，监控账户将能够查看源账户中的数据。您可以使用接收器来创建从源账户指向监控账户的链接。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

所需的 资源

为保证 CloudWatch Application Insights 跨账户可观测性正常运行，请务必通过 CloudWatch Observability Access Manager 共享以下遥测类型。

- CloudWatch Application Insights 中的应用程序
- Amazon CloudWatch 中的指标
- 在 Amazon CloudWatch Logs 中记录组
- AWS X-Ray 中的跟踪记录

使用组件配置

组件配置是 JSON 格式的文本文件，用于描述组件的配置设置。本节提供示例模板片段、组件配置部分的说明以及组件配置示例。

主题

- 组件配置模板片段
- 组件配置部分
- 组件配置示例

组件配置模板片段

以下示例显示 JSON 格式的模板片段。

```
{
  "alarmMetrics" : [
    list of alarm metrics
  ],
  "logs" : [
    list of logs
  ],
  "processes" : [
   list of processes
  ],
  "windowsEvents" : [
    list of windows events channels configurations
  ],
  "alarms" : [
    list of CloudWatch alarms
  ],
  "jmxPrometheusExporter": {
```

```
    JMX Prometheus Exporter configuration
  },
  "hanaPrometheusExporter": {
      SAP HANA Prometheus Exporter configuration
  },
  "haClusterPrometheusExporter": {
      HA Cluster Prometheus Exporter configuration
  },
  "netWeaverPrometheusExporter": {
      SAP NetWeaver Prometheus Exporter configuration
  },
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance" ...
      component nested instances configuration
    },
    {
      "subComponentType" : "AWS::EC2::Volume" ...
      component nested volumes configuration
    }
  ]
}
```

组件配置部分

组件配置包括几个主要部分。组件配置的部分可以任何顺序列出。

- alarmMetrics（可选）

  组件中要监控的指标列表。所有组件类型都可以有一个 alarmMetrics 部分。

- 日志（可选）

  组件中要监控的日志列表。仅 EC2 实例可以具有日志部分。

- 进程（可选）

  要为组件监控的进程的列表。仅 EC2 实例可以具有进程部分。

- subComponents（可选）

  组件的嵌套实例和卷子组件配置。以下类型的组件可以具有嵌套实例和一个子组件部分：ELB、ASG、自定义分组的 EC2 实例和 EC2 实例。

- 警报（可选）

要为组件监控的警报的列表。所有组件类型都可以有一个警报部分。

- windowsEvents（可选）

  要为组件监控的 Windows 事件的列表。只有 EC2 实例上的 Windows 有 `windowsEvents` 部分。

- JMXPrometheusExporter（可选）

  JMXPrometheus Exporter 配置。

- hanaPrometheusExporter（可选）

  SAP HANA Prometheus Exporter 配置。

- haClusterPrometheusExporter（可选）

  HA Cluster Prometheus Exporter 配置。

- netWeaverPrometheusExporter（可选）

  SAP NetWeaver Prometheus Exporter 配置。

- sapAsePrometheusExporter（可选）

  SAP ASE Prometheus Exporter 配置。

以下示例显示 JSON 格式的 subComponents 部分片段的语法。

```
[
  {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
      list of alarm metrics
    ],
    "logs" : [
      list of logs
    ],
    "processes": [
      list of processes
    ],
    "windowsEvents" : [
      list of windows events channels configurations
    ]
  },
  {
```

```
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      list of alarm metrics
    ]
  }
]
```

组件配置部分属性

这部分描述各部件配置部分的属性。

Sections

- 指标
- Log
- 过程
- JMX Prometheus Exporter
- HANA Prometheus Exporter
- HA Cluster Prometheus Exporter
- NetWeaver Prometheus Exporter
- SAP ASE Prometheus Exporter
- Windows 事件
- 警报

指标

定义组件中要监控的指标。

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

属性

- alarmMetricName（必需）

组件中待监控指标的名称。有关 Application Insights 支持的指标的信息，请参阅 Amazon CloudWatch Application Insights 支持的日志和指标。

- 监控（可选）

  布尔值，指示是否监控指标。默认值为 `true`。

Log

定义组件中要监控的日志。

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

属性

- logGroupName（必需）

  要与监控日志关联的 CloudWatch 日志组名称。对于日志组名称约束的信息，请参阅 CreateLogGroup。

- logPath（EC2 实例组件需要；不使用 CloudWatch 代理的组件不需要，例如 AWS Lambda）

  要监控的日志路径。日志路径必须是 Windows 系统文件绝对路径。有关更多信息，请参阅 CloudWatch 代理配置文件：日志部分。

- logType（必需）

  日志类型决定 Application Insights 分析日志时所依据的日志模式。从以下选项中选择日志类型：

  - `SQL_SERVER`
  - `MYSQL`
  - `MYSQL_SLOW_QUERY`
  - `POSTGRESQL`

  - ORACLE_ALERT
  - ORACLE_LISTENER
  - IIS
  - APPLICATION
  - WINDOWS_EVENTS
  - WINDOWS_EVENTS_ACTIVE_DIRECTORY
  - WINDOWS_EVENTS_DNS
  - WINDOWS_EVENTS_IIS
  - WINDOWS_EVENTS_SHAREPOINT
  - SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP
  - SQL_SERVER_FAILOVER_CLUSTER_INSTANCE
  - DEFAULT
  - CUSTOM
  - STEP_FUNCTION
  - API_GATEWAY_ACCESS
  - API_GATEWAY_EXECUTION
  - SAP_HANA_LOGS
  - SAP_HANA_TRACE
  - SAP_HANA_HIGH_AVAILABILITY
  - SAP_NETWEAVER_DEV_TRACE_LOGS
  - PACEMAKER_HIGH_AVAILABILITY
- 编码（可选）

  要监控的日志的编码类型。指定的编码应包含在 CloudWatch 代理支持的编码列表中。如果未提供，CloudWatch Application Insights 将对 utf-8 类型使用默认编码，以下除外：

  - SQL_SERVER：utf-16 编码
  - IIS：ascii 编码
- 监控（可选）

  布尔值，用于指示是否监控日志。默认值为 true。

过程

定义组件待监控的过程。

JSON

```
{
  "processName" : "monitoredProcessName",
  "alarmMetrics" : [
      list of alarm metrics
  ]
}
```

属性

- processName（必需）

  组件待监控的过程名称。进程名称不能包含进程主干，如 `sqlservr` 或 `sqlservr.exe`。
- alarmMetrics（必需）

  此进程要监控的指标的列表。要查看 CloudWatch Application Insights 支持的进程指标，请参阅 Amazon Elastic Compute Cloud (EC2) 。

JMX Prometheus Exporter

定义 JMX Prometheus Exporter 设置。

JSON

```
"JMXPrometheusExporter": {
  "jmxURL" : "JMX URL",
  "hostPort" : "The host and port",
  "prometheusPort" : "Target port to emit Prometheus metrics"
}
```

属性

- jmxURL（可选）

  要连接到的完整 JMX URL。
- hostPort（可选）

要通过远程 JMX 连接的主机和端口。只能指定 jmxURL 和 hostPort 中的一个。

- prometheusPort（可选）

  要向其发送 Prometheus 指标的目标端口。如果未指定，则使用默认端口 9404。

HANA Prometheus Exporter

定义 HANA Prometheus Exporter 设置。

JSON

```
"hanaPrometheusExporter": {
    "hanaSid": "SAP HANA  SID",
    "hanaPort": "HANA database port",
    "hanaSecretName": "HANA secret name",
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

属性

- hanaSid

  SAP HANA 系统由三个字符组成的 SAP 系统 ID (SID)。

- hanaPort

  导出程序将用于查询 HANA 指标的 HANA 数据库端口。

- HanaSecretName

  存储 HANA 监控用户凭证的 AWS Secrets Manager 密钥。HANA Prometheus Exporter 使用这些凭证连接到数据库并查询 HANA 指标。

- prometheusPort（可选）

  Prometheus 向其发送指标的目标端口。如果未指定，则使用默认端口 9668。

HA Cluster Prometheus Exporter

定义 HA Cluster Prometheus Exporter 设置。

JSON

```
"haClusterPrometheusExporter": {
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

属性

- prometheusPort（可选）

  Prometheus 向其发送指标的目标端口。如果未指定，则使用默认端口 9664。

NetWeaver Prometheus Exporter

定义 NetWeaver Prometheus Exporter 设置。

JSON

```
"netWeaverPrometheusExporter": {
    "sapSid": "SAP NetWeaver  SID",
    "instanceNumbers": [ "Array of instance Numbers of SAP NetWeaver system "],
"prometheusPort": "Target port to emit Prometheus metrics"
}
```

属性

- sapSid

  SAP NetWeaver 系统的 3 字符 SAP 系统 ID (SID)。
- instanceNumbers

  SAP NetWeaver 系统的实例号数组。

  示例：`"instanceNumbers": [ "00", "01"]`
- prometheusPort（可选）

  发送 Prometheus 指标的目标端口。如果未指定，则使用默认端口 9680。

SAP ASE Prometheus Exporter

定义 SAP ASE Prometheus Exporter 设置。

JSON

```
"sapASEPrometheusExporter": {
    "sapAseSid": "SAP ASE SID",
    "sapAsePort": "SAP ASE database port",
    "sapAseSecretName": "SAP ASE secret name",
    "prometheusPort": "Target port to emit Prometheus metrics",
    "agreeToEnableASEMonitoring": true
}
```

属性

- sapAseSid

  SAP ASE 系统由三个字符组成的 SAP 系统 ID（SID）。

- sapAsePort

  导出程序将用于查询 ASE 指标的 SAP ASE 数据库端口。

- sapAseSecretName

  存储 ASE 监控用户凭证的 AWS Secrets Manager 密钥。SAP ASE Prometheus 导出程序使用这些凭证连接到数据库并查询 ASE 指标。

- prometheusPort（可选）

  Prometheus 向其发送指标的目标端口。如果未指定，则使用默认端口 9399。如果有另一个 ASE 数据库使用默认端口，则我们使用端口 9499。

Windows 事件

定义要记录的 Windows 事件。

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR","WARNING","CRITICAL","INFORMATION","VERBOSE"],
  "monitor" : true/false
}
```

属性

- logGroupName（必需）

  要与监控日志关联的 CloudWatch 日志组名称。对于日志组名称约束的信息，请参阅 CreateLogGroup。

- eventName（必需）

  要记录的 Windows 事件的类型。这等同于 Windows 事件日志通道名称。例如，系统、安全、CustomEventName 等。要记录的每种类型的 Windows 事件需要使用该字段。

- eventLevels（必需）

  要记录的事件级别。您必须指定要录入的每个级别。可能的值包括 `INFORMATION`、`WARNING`、`ERROR`、`CRITICAL` 和 `VERBOSE`。要记录的每种类型的 Windows 事件需要使用该字段。

- 监控（可选）

  布尔值，用于指示是否监控日志。默认值为 `true`。

警报

定义要为组件监控的 CloudWatch 告警。

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

属性

- alarmName（必需）

  要为组件监控的 CloudWatch 告警的名称。

- 严重性（可选）

  指示告警关闭时的中断程度。

组件配置示例

以下示例以 JSON 格式显示了相关服务的组件配置。

示例组件配置

- Amazon DynamoDB 表
- Amazon EC2 Auto Scaling (ASG)
- Amazon EKS 集群
- Amazon Elastic Compute Cloud (EC2) 实例
- Amazon Elastic Container Service (Amazon ECS)
- Amazon ECS 服务
- Amazon ECS 任务
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx
- Amazon Relational Database Service (RDS) Aurora MySQL
- Amazon Relational Database Service（RDS）实例
- Amazon Route 53 运行状况检查
- Amazon Route 53 托管区
- Amazon Route 53 Resolver 端点
- Amazon Route 53 Resolver 查询日志记录配置
- Amazon S3 存储桶
- Amazon Simple Queue Service (SQS)
- Amazon SNS 主题
- Amazon Virtual Private Cloud (Amazon VPC)
- Amazon VPC 网络地址转换（NAT）网关
- API Gateway REST API 阶段
- Application Elastic Load Balancing
- AWS Lambda 函数
- AWS Network Firewall 规则组
- AWS Network Firewall 规则组关联
- AWS Step Functions
- 客户分组的 Amazon EC2 实例

- Elastic Load Balancing
- Java
- Amazon EC2 上的 Kubernetes
- RDS MariaDB 和 RDS MySQL
- RDS Oracle
- RDS PostgreSQL
- Amazon EC2 上的 SAP ASE
- Amazon EC2 上的 SAP ASE 高可用性
- Amazon EC2 上的 SAP HANA
- Amazon EC2 上的 SAP HANA 高可用性
- Amazon EC2 上的 SAP NetWeaver
- Amazon EC2 上的 SAP NetWeaver 高可用性
- SQL Always On 可用性组
- SQL 故障转移集群实例

Amazon DynamoDB 表

以下示例演示适用于 Amazon DynamoDB 表的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "SystemErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "UserErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedReadCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedWriteCapacityUnits",
      "monitor": false
    },
```

```
    {
      "alarmMetricName": "ReadThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "WriteThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "ConditionalCheckFailedRequests",
      "monitor": false
    },
    {
      "alarmMetricName": "TransactionConflict",
      "monitor": false
    }
  ],
  "logs": []
}
```

Amazon EC2 Auto Scaling (ASG)

以下示例演示适用于 Amazon EC2 Auto Scaling (ASG) 的 JSON 格式组件配置。

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUCreditBalance"
      }, {
        "alarmMetricName" : "EBSIOBalance%"
      }
    ],
    "subComponents" : [
      {
        "subComponentType" : "AWS::EC2::Instance",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "CPUUtilization"
          }, {
            "alarmMetricName" : "StatusCheckFailed"
          }
        ],
        "logs" : [
          {
```

```
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION"
          }
        ],
        "processes" : [
          {
            "processName" : "my_process",
            "alarmMetrics" : [
              {
                  "alarmMetricName" : "procstat cpu_usage",
                  "monitor" : true
              }, {
                  "alarmMetricName" : "procstat memory_rss",
                  "monitor" : true
              }
          ]
      }
  ],
        "windowsEvents" : [
          {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType" : "AWS::EC2::Volume",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "VolumeQueueLength"
          }, {
            "alarmMetricName" : "BurstBalance"
          }
        ]
      }
    ],
    "alarms" : [
      {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
      }
    ]
```

```
  }
```

Amazon EKS 集群

以下示例演示适用于 Amazon EKS 集群的 JSON 格式组件配置。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName": "cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "node_memory_utilization",
          "monitor":true
       },
       {
          "alarmMetricName": "node_network_total_bytes",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName": "pod_cpu_utilization",
          "monitor":true
       },
```

```
        {
           "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName": "pod_memory_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName": "pod_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName": "pod_memory_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName": "pod_network_rx_bytes",
           "monitor":true
        },
        {
           "alarmMetricName": "pod_network_tx_bytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName": "/aws/containerinsights/kubernetes/application",
           "logType":"APPLICATION",
           "monitor":true,
           "encoding":"utf-8"
        }
     ],
     "subComponents":[
        {
           "subComponentType":"AWS::EC2::Instance",
           "alarmMetrics":[
              {
                 "alarmMetricName":"CPUUtilization",
                 "monitor":true
              },
              {
                 "alarmMetricName":"StatusCheckFailed",
                 "monitor":true
```

```
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
               "logPath":"",
               "logType":"APPLICATION",
               "monitor":true,
               "encoding":"utf-8"
            }
         ],
         "processes" : [
           {
               "processName" : "my_process",
               "alarmMetrics" : [
                   {
                       "alarmMetricName" : "procstat cpu_usage",
                       "monitor" : true
                   }, {
                       "alarmMetricName" : "procstat memory_rss",
                       "monitor" : true
                   }
               ]
           }
      ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
```

```
            ]
        },
        {
            "subComponentType":"AWS::AutoScaling::AutoScalingGroup",
            "alarmMetrics":[
                {
                    "alarmMetricName":"CPUCreditBalance",
                    "monitor":true
                },
                {
                    "alarmMetricName":"EBSIOBalance%",
                    "monitor":true
                }
            ]
        },
        {
            "subComponentType":"AWS::EC2::Volume",
            "alarmMetrics":[
                {
                    "alarmMetricName":"VolumeReadBytes",
                    "monitor":true
                },
                {
                    "alarmMetricName":"VolumeWriteBytes",
                    "monitor":true
                },
                {
                    "alarmMetricName":"VolumeReadOps",
                    "monitor":true
                },
                {
                    "alarmMetricName":"VolumeWriteOps",
                    "monitor":true
                },
                {
                    "alarmMetricName":"VolumeQueueLength",
                    "monitor":true
                },
                {
                    "alarmMetricName":"BurstBalance",
                    "monitor":true
                }
            ]
        }
```

```
    ]
}
```

> **Note**
>
> - `AWS::EC2::Instance`、`AWS::EC2::Volume` 和 `AWS::AutoScaling::AutoScalingGroup` 的 `subComponents` 部分仅适用于在 EC2 启动类型上运行的 Amazon EKS 集群。
> - `AWS::EC2::Instance` 在 `subComponents` 的 `windowsEvents` 部分仅适用于 Amazon EC2 实例上运行的 Windows。

Amazon Elastic Compute Cloud (EC2) 实例

以下示例演示适用于 Amazon EC2 实例的 JSON 格式组件配置。

> **Important**
>
> 当 Amazon EC2 实例进入 `stopped` 状态时，表示已将它从监控中移除。当它恢复为 `running` 状态时，会将它添加到 CloudWatch Application Insights 控制台 Application details（应用程序详细信息）页面上的 Unmonitored components（未监控组件）列表中。如果为应用程序启用了自动监控新资源，则会将该实例添加到 Monitored components（已监控组件）列表中。但会将日志和指标设置为工作负载的默认值。不会保存之前的日志和指标配置。

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUUtilization",
        "monitor" : true
      }, {
        "alarmMetricName" : "StatusCheckFailed"
      }
    ],
    "logs" : [
      {
        "logGroupName" : "my_log_group",
        "logPath" : "C:\\LogFolder\\*",
        "logType" : "APPLICATION",
```

```
        "monitor" : true
      },
      {
        "logGroupName" : "my_log_group_2",
        "logPath" : "C:\\LogFolder2\\*",
        "logType" : "IIS",
        "encoding" : "utf-8"
      }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ],
    "windowsEvents" : [
      {
        "logGroupName" : "my_log_group_3",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
      }, {
        "logGroupName" : "my_log_group_4",
        "eventName" : "System",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
    }],
     "alarms" : [
      {
        "alarmName" : "my_instance_alarm_1",
        "severity" : "HIGH"
      },
      {
        "alarmName" : "my_instance_alarm_2",
        "severity" : "LOW"
      }
```

```
    ],
     "subComponents" : [
     {
       "subComponentType" : "AWS::EC2::Volume",
       "alarmMetrics" : [
       {
         "alarmMetricName" : "VolumeQueueLength",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "VolumeThroughputPercentage",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "BurstBalance",
         "monitor" : "true"
       }
    }]
  }
```

Amazon Elastic Container Service (Amazon ECS)

以下示例演示适用于 Amazon Elastic Container Service (Amazon ECS) 的 JSON 格式组件配置。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkRxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkTxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"RunningTaskCount",
```

```
            "monitor":true
         },
         {
            "alarmMetricName":"PendingTaskCount",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageReadBytes",
            "monitor":true
         },
         {
            "alarmMetricName":"StorageWriteBytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/ecs/my-task-definition",
            "logType":"APPLICATION",
            "monitor":true
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Backend_4XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"HTTPCode_Backend_5XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"Latency",
                  "monitor":true
               },
               {
                  "alarmMetricName":"SurgeQueueLength",
                  "monitor":true
               },
               {
                  "alarmMetricName":"UnHealthyHostCount",
```

```
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
        "alarmMetrics":[
            {
                "alarmMetricName":"HTTPCode_Target_4XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"HTTPCode_Target_5XX_Count",
                "monitor":true
            },
            {
                "alarmMetricName":"TargetResponseTime",
                "monitor":true
            },
            {
                "alarmMetricName":"UnHealthyHostCount",
                "monitor":true
            }
        ]
    },
    {
        "subComponentType":"AWS::EC2::Instance",
        "alarmMetrics":[
            {
                "alarmMetricName":"CPUUtilization",
                "monitor":true
            },
            {
                "alarmMetricName":"StatusCheckFailed",
                "monitor":true
            },
            {
                "alarmMetricName":"disk_used_percent",
                "monitor":true
            },
            {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
            }
```

```
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
               "processName" : "my_process",
               "alarmMetrics" : [
                   {
                       "alarmMetricName" : "procstat cpu_usage",
                       "monitor" : true
                   }, {
                       "alarmMetricName" : "procstat memory_rss",
                       "monitor" : true
                   }
                ]
            }
         ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":"true"
            },
            {
```

```
                "alarmMetricName":"VolumeThroughputPercentage",
                "monitor":"true"
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":"true"
             }
          ]
       }
    ]
}
```

> **Note**
>
> - `AWS::EC2::Instance` 和 `AWS::EC2::Volume` 的 `subComponents` 部分仅适用于 ECS 服务或 ECS 任务在 EC2 启动类型上运行的 Amazon ECS 集群。
> - `AWS::EC2::Instance` 在 `subComponents` 的 `windowsEvents` 部分仅适用于 Amazon EC2 实例上运行的 Windows。

Amazon ECS 服务

以下示例演示了一个 JSON 格式的 Amazon ECS 服务组件配置。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CPUUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
```

```
        },
        {
           "alarmMetricName":"NetworkRxBytes",
           "monitor":true
        },
        {
           "alarmMetricName":"NetworkTxBytes",
           "monitor":true
        },
        {
           "alarmMetricName":"RunningTaskCount",
           "monitor":true
        },
        {
           "alarmMetricName":"PendingTaskCount",
           "monitor":true
        },
        {
           "alarmMetricName":"StorageReadBytes",
           "monitor":true
        },
        {
           "alarmMetricName":"StorageWriteBytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName":"/ecs/my-task-definition",
           "logType":"APPLICATION",
           "monitor":true
        }
     ],
     "subComponents":[
        {
           "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
           "alarmMetrics":[
              {
                 "alarmMetricName":"HTTPCode_Backend_4XX",
                 "monitor":true
              },
              {
                 "alarmMetricName":"HTTPCode_Backend_5XX",
                 "monitor":true
```

```
            },
            {
               "alarmMetricName":"Latency",
               "monitor":true
            },
            {
               "alarmMetricName":"SurgeQueueLength",
               "monitor":true
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
         "alarmMetrics":[
            {
               "alarmMetricName":"HTTPCode_Target_4XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"HTTPCode_Target_5XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"TargetResponseTime",
               "monitor":true
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Instance",
         "alarmMetrics":[
            {
               "alarmMetricName":"CPUUtilization",
               "monitor":true
            },
            {
```

```
                "alarmMetricName":"StatusCheckFailed",
                "monitor":true
            },
            {
                "alarmMetricName":"disk_used_percent",
                "monitor":true
            },
            {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
            }
        ],
        "logs":[
            {
                "logGroupName":"my_log_group",
                "logPath":"/mylog/path",
                "logType":"APPLICATION",
                "monitor":true
            }
        ],
        "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ],
        "windowsEvents":[
            {
                "logGroupName":"my_log_group_2",
                "eventName":"Application",
                "eventLevels":[
                    "ERROR",
                    "WARNING",
                    "CRITICAL"
                ],
                "monitor":true
```

```
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":"true"
            },
            {
               "alarmMetricName":"VolumeThroughputPercentage",
               "monitor":"true"
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":"true"
            }
         ]
      }
   ]
}
```

> **Note**
>
> - `AWS::EC2::Instance` 和 `AWS::EC2::Volume` 的 `subComponents` 部分仅适用于在 EC2 启动类型上运行的 Amazon ECS。
> - `AWS::EC2::Instance` 在 `subComponents` 的 `windowsEvents` 部分仅适用于 Amazon EC2 实例上运行的 Windows。

Amazon ECS 任务

以下示例演示了一个 JSON 格式的 Amazon ECS 任务组件配置。

```
{
    "logs":[
       {
          "logGroupName":"/ecs/my-task-definition",
          "logType":"APPLICATION",
          "monitor":true
```

```
        }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ]
}
```

Amazon Elastic File System (Amazon EFS)

以下示例演示了一个 JSON 格式的 Amazon EFS 组件配置。

```
{
   "alarmMetrics": [
     {
       "alarmMetricName": "BurstCreditBalance",
       "monitor": true
     },
     {
       "alarmMetricName": "PercentIOLimit",
       "monitor": true
     },
     {
       "alarmMetricName": "PermittedThroughput",
       "monitor": true
     },
     {
       "alarmMetricName": "MeteredIOBytes",
       "monitor": true
     },
     {
       "alarmMetricName": "TotalIOBytes",
       "monitor": true
     },
```

```
    {
      "alarmMetricName": "DataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "ClientConnections",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeSinceLastSync",
      "monitor": true
    },
    {
      "alarmMetricName": "Throughput",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfPermittedThroughputUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "ThroughputIOPS",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfIOPSDataReadIOBytes",
      "monitor": true
    },
```

```
    {
      "alarmMetricName": "PercentageOfIOPSDataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "AverageDataReadIOBytesSize",
      "monitor": true
    },
    {
      "alarmMetricName": "AverageDataWriteIOBytesSize",
      "monitor": true
    }
  ],
  "logs": [
   {
   "logGroupName": "/aws/efs/utils",
   "logType": "EFS_MOUNT_STATUS",
   "monitor": true,
   }
  ]
}
```

Amazon FSx

以下示例演示适用于 Amazon FSx 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteOperations",
      "monitor": true
    },
```

```
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

以下示例演示适用于 Amazon RDS Aurora MySQL 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

Amazon Relational Database Service（RDS）实例

以下示例演示适用于 Amazon RDS 实例的 JSON 格式组件配置。

```
{
```

```
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    }, {
      "alarmMetricName" : "WriteThroughput",
      "monitor" : false
    }
  ],

  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Route 53 运行状况检查

以下示例演示了一个 JSON 格式的 Amazon Route 53 运行状况检查。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ChildHealthCheckHealthyCount",
      "monitor": true
    },
    {
      "alarmMetricName": "ConnectionTime",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckPercentageHealthy",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckStatus",
      "monitor": true
    },
    {
      "alarmMetricName": "SSLHandshakeTime",
      "monitor": true
    },
```

```
    {
      "alarmMetricName": "TimeToFirstByte",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 托管区

以下示例演示了一个 JSON 格式的 Amazon Route 53 托管区。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DNSQueries",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECInternalFailure",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeysNeedingAction",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyMaxNeedingActionAge",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyAge",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName":"/hosted-zone/logs",
      "logType": "ROUTE53_DNS_PUBLIC_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver 端点

以下示例演示适用于 Amazon Route 53 Resolver 端点的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "EndpointHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "EndpointUnHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "InboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryAggregateVolume",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver 查询日志记录配置

下面的例子演示了适用于 Amazon Route 53 Resolver 查询日志记录配置的 JSON 格式的组件配置。

```
{
  "logs": [
    {
      "logGroupName": "/resolver-query-log-config/logs",
      "logType": "ROUTE53_RESOLVER_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon S3 存储桶

以下示例演示适用于 Amazon S3 存储桶的 JSON 格式组件配置。

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "ReplicationLatency",
            "monitor" : true
        }, {
            "alarmMetricName" : "5xxErrors",
            "monitor" : true
        }, {
            "alarmMetricName" : "BytesDownloaded"
            "monitor" : true
        }
    ]
}
```

Amazon Simple Queue Service (SQS)

以下示例演示适用于 Amazon Simple Queue Service 的 JSON 格式组件配置。

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ApproximateAgeOfOldestMessage"
    }, {
      "alarmMetricName" : "NumberOfEmptyReceives"
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_sqs_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon SNS 主题

以下示例演示适用于 Amazon SNS 主题的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

Amazon Virtual Private Cloud (Amazon VPC)

以下示例演示了适用于 Amazon VPC 的 JSON 格式的组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NetworkAddressUsage",
      "monitor": true
    },
    {
      "alarmMetricName": "NetworkAddressUsagePeered",
      "monitor": true
    },
    {
      "alarmMetricName": "VPCFirewallQueryVolume",
      "monitor": true
    }
  ]
}
```

Amazon VPC 网络地址转换（NAT）网关

以下示例演示了适用于 NAT 网关的 JSON 格式的组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ErrorPortAllocation",
      "monitor": true
    },
    {
      "alarmMetricName": "IdleTimeoutCount",
      "monitor": true
    }
  ]
}
```

API Gateway REST API 阶段

以下示例演示适用于 API Gateway REST API 阶段的 JSON 格式组件配置。

```
{
     "alarmMetrics" : [
         {
             "alarmMetricName" : "4XXError",
             "monitor" : true
         },
         {
             "alarmMetricName" : "5XXError",
             "monitor" : true
         }
     ],
    "logs" : [
        {
            "logType" : "API_GATEWAY_EXECUTION",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_ACCESS",
            "monitor" : true
        }
    ]
}
```

Application Elastic Load Balancing

以下示例演示适用于 Application Elastic Load Balancing 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ActiveConnectionCount",
    }, {
      "alarmMetricName": "TargetResponseTime"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ],
      "logs": [
        {
          "logGroupName": "my_log_group",
          "logPath": "C:\\LogFolder\\*",
          "logType": "APPLICATION",
        }
      ],
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
        }
      ]
    }, {
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength",
        }, {
          "alarmMetricName": "BurstBalance"
        }
```

```
      ]
    }
  ],

  "alarms": [
    {
      "alarmName": "my_alb_alarm",
      "severity": "LOW"
    }
  ]
}
```

AWS Lambda 函数

以下示例演示适用于 AWS Lambda Function 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
      "alarmMetricName": "IteratorAge",
      "monitor": true
    },
    {
      "alarmMetricName": "Duration",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "DEFAULT",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall 规则组

以下示例演示了适用于 AWS Network Firewall 规则组的 JSON 格式的组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall 规则组关联

以下示例演示了适用于 AWS Network Firewall 规则组关联的 JSON 格式的组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Step Functions

以下示例演示适用于 AWS Step Functions 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ExecutionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "LambdaFunctionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "ProvisionedRefillRate",
      "monitor": true
```

```
    }
  ],
  "logs": [
    {
     "logGroupName": "/aws/states/HelloWorld-Logs",
      "logType": "STEP_FUNCTION",
      "monitor": true,
    }
  ]
}
```

客户分组的 Amazon EC2 实例

以下示例演示适用于客户分组的 Amazon EC2 实例的 JSON 格式组件配置。

```
{
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization",
          },
          {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION",
          }
        ],
        "processes": [
            {
                "processName": "my_process",
                "alarmMetrics": [
                    {
                        "alarmMetricName": "procstat cpu_usage",
                        "monitor": true
                    }, {
                        "alarmMetricName": "procstat memory_rss",
                        "monitor": true
```

```
                    }
                ]
            }
        ],
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength",
          }, {
            "alarmMetricName": "BurstBalance"
          }
        ]
      }
    ],
    "alarms": [
      {
        "alarmName": "my_alarm",
        "severity": "MEDIUM"
      }
    ]
  }
```

Elastic Load Balancing

以下示例演示适用于 Elastic Load Balancing 的 JSON 格式组件配置。

```
{
    "alarmMetrics": [
      {
        "alarmMetricName": "EstimatedALBActiveConnectionCount"
      }, {
        "alarmMetricName": "HTTPCode_Backend_5XX"
      }
    ],
    "subComponents": [
      {
```

```
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization"
          }, {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION"
          }
        ],
        "processes": [
          {
            "processName": "my_process",
            "alarmMetrics": [
                {
                  "alarmMetricName": "procstat cpu_usage",
                  "monitor": true
                }, {
                  "alarmMetricName": "procstat memory_rss",
                  "monitor": true
                }
             ]
          }
        ],
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ],
            "monitor": true
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength"
          }, {
            "alarmMetricName": "BurstBalance"
```

```
          }
        ]
      }
    ],

    "alarms": [
      {
        "alarmName": "my_elb_alarm",
        "severity": "HIGH"
      }
    ]
  }
```

Java

以下示例演示适用于 Java 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [ {
    "alarmMetricName": "java_lang_threading_threadcount",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_used",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_committed",
    "monitor": true
  }],
  "logs": [ ],
  "JMXPrometheusExporter": {
      "hostPort": "8686",
      "prometheusPort": "9404"
  }
}
```

> **Note**
>
> Application Insights 不支持为 Prometheus JMX Exporter 配置身份验证。有关如何设置身份验证的信息，请参阅 Prometheus JMX Exporter 示例配置。

Amazon EC2 上的 Kubernetes

以下示例演示适用于 Amazon EC2 上 Kubernetes 的 JSON 格式组件配置。

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_memory_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_memory_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_network_total_bytes",
          "monitor":true
       },
       {
          "alarmMetricName":"pod_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"pod_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"pod_cpu_utilization_over_pod_limit",
```

```
            "monitor":true
         },
         {
            "alarmMetricName":"pod_memory_reserved_capacity",
            "monitor":true
         },
         {
            "alarmMetricName":"pod_memory_utilization",
            "monitor":true
         },
         {
            "alarmMetricName":"pod_memory_utilization_over_pod_limit",
            "monitor":true
         },
         {
            "alarmMetricName":"pod_network_rx_bytes",
            "monitor":true
         },
         {
            "alarmMetricName":"pod_network_tx_bytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/aws/containerinsights/kubernetes/application",
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::EC2::Instance",
            "alarmMetrics":[
               {
                  "alarmMetricName":"CPUUtilization",
                  "monitor":true
               },
               {
                  "alarmMetricName":"StatusCheckFailed",
                  "monitor":true
               },
               {
```

```
                "alarmMetricName":"disk_used_percent",
                "monitor":true
             },
             {
                "alarmMetricName":"mem_used_percent",
                "monitor":true
             }
          ],
          "logs":[
             {
                "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                "logPath":"",
                "logType":"APPLICATION",
                "monitor":true,
                "encoding":"utf-8"
             }
          ],
          "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                ]
            }
       ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
             {
                "alarmMetricName":"VolumeReadBytes",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteBytes",
                "monitor":true
             },
             {
```

```
                "alarmMetricName":"VolumeReadOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeQueueLength",
                "monitor":true
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":true
             }
          ]
       }
    ]
 }
```

RDS MariaDB 和 RDS MySQL

以下示例演示适用于 RDS MariaDB 和 RDS MySQL 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS Oracle

以下示例演示适用于 RDS Oracle 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "ORACLE_ALERT",
      "monitor": true,
    },
    {
      "logType": "ORACLE_LISTENER",
      "monitor": false
    }
  ]
}
```

RDS PostgreSQL

以下示例演示适用于 RDS PostgreSQL 的 JSON 格式组件配置。

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

Amazon EC2 上的 SAP ASE

以下示例显示了适用于 Amazon EC2 上 SAP ASE 的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_log_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_locked_login",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_data_cache_hit_ratio",
          "monitor": true
```

```
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

Amazon EC2 上的 SAP ASE 高可用性

以下示例显示了适用于 Amazon EC2 上 SAP ASE 高可用性的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
```

```
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_mode",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_latency_in_minutes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
```

```
          "logGroupName": "SAP_ASE_REP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/repservername/repservername.log",
          "logType": "SAP_ASE_REP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_RMA_AGENT_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/RMA-*/instances/AgentContainer/logs/",
          "logType": "SAP_ASE_RMA_AGENT_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_FAULT_MANAGER_LOGS-my-resource-group",
          "logPath": "/opt/sap/FaultManager/dev_sybdbfm",
          "logType": "SAP_ASE_FAULT_MANAGER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

Amazon EC2 上的 SAP HANA

以下示例演示适用于 Amazon EC2 上 SAP HANA 的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_table_allocation_limit_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_cpu_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_last_data_backup_age_days",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_LOGS-my-resource-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",
          "logType": "SAP_HANA_LOGS",
          "monitor": true,
```

```
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  }
}
```

Amazon EC2 上的 SAP HANA 高可用性

以下示例演示适用于 Amazon EC2 上 SAP HANA 高可用性的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
          "monitor": true
        }
      ],
      "logs": [
```

```
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
          "logPath": "/var/log/pacemaker/pacemaker.log",
          "logType": "SAP_HANA_HIGH_AVAILABILITY",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  },
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  }
}
```

Amazon EC2 上的 SAP NetWeaver

以下示例显示了适用于 Amazon EC2 上的 SAP NetWeaver 的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
          "monitor": true
        },
        {
          "alarmMetricName": "StatusCheckFailed",
          "monitor": true
```

```
        },
        {
          "alarmMetricName": "disk_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "mem_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialog",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_DBRequestTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_LongRunners",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_BasisSystem",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Database",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Security",
          "monitor": true
```

```
        },
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_QueueTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_max",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_max",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        }
      ],
      "logs": [
        {
```

```
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-ML4",
          "logPath": "/usr/sap/ML4/*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "netWeaverPrometheusExporter": {
    "sapSid": "ML4",
    "instanceNumbers": [
      "00",
      "11"
    ],
    "prometheusPort": "9680"
  }
}
```

Amazon EC2 上的 SAP NetWeaver 高可用性

以下示例显示了适用于 Amazon EC2 上的 SAP NetWeaver 高可用性的 JSON 格式组件配置。

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "ha_cluster_corosync_ring_errors",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_fail_count",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_check_failover_config_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_get_failover_config_HAActive",
          "monitor": true
        },
```

```
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_BasisSystem",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_DBRequestTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Database",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_FrontendResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_LongRunners",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_QueueTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialog",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
          "monitor": true
        },
```

```
        {
          "alarmMetricName": "sap_alerts_Security",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Shortdumps",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_SqlError",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-PR1",
          "logPath": "/usr/sap/<SID>/D*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  },
  "netWeaverPrometheusExporter": {
    "sapSid": "PR1",
    "instanceNumbers": [
      "11",
      "12"
```

```
    ],
    "prometheusPort": "9680"
  }
}
```

SQL Always On 可用性组

以下示例演示适用于 SQL Always On 可用性组的 JSON 格式组件配置。

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
```

```
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
```

```
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
```

```
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

SQL 故障转移集群实例

以下示例演示适用于 SQL 故障转移集群实例的 JSON 格式组件配置。

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Bytes Received/sec",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "Normal Messages Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Urgent Message Queue Length/se",
      "monitor" : true
    }, {
      "alarmMetricName" : "Reconnect Count",
      "monitor" : true
    }, {
      "alarmMetricName" : "Unacknowledged Message Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Outstanding",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Sent/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Database Update Messages/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Update Messages/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Flushes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster API Calls/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource API Calls/sec",
```

```
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster Handles/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource Handles/sec",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL"],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
      "logPath" : "\\\\amznfsxjmzbykwn.mydomain.aws\\SQLDB\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
```

```
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

使用 CloudFormation 模板创建和配置 CloudWatch Application Insights 监控

您可以直接从 AWS CloudFormation 模板中将 Application Insights 监控（包括关键指标和遥测）添加到应用程序、数据库和 Web 服务器。

此部分提供 JSON 和 YAML 格式的示例 AWS CloudFormation 模板，以帮助您创建和配置 Application Insights 监控。

要查看 AWS CloudFormation 用户指南中的 Application Insights 资源和属性参考，请参阅 ApplicationInsights 资源类型参考。

示例 模板

- 为整个 AWS CloudFormation 堆栈创建 Application Insights 应用程序
- 创建具有详细设置的 Application Insights 应用程序
- 使用 CUSTOM 模式组件配置创建 Application Insights 应用程序
- 使用 DEFAULT 模式组件配置创建 Application Insights 应用程序
- 使用 DEFAULT_WITH_OVERWRITE 模式组件配置创建 Application Insights 应用程序

为整个 AWS CloudFormation 堆栈创建 Application Insights 应用程序

要应用以下模板，必须创建 AWS 资源和一个或多个资源组，用于创建 Application Insights 应用程序以监控这些资源。有关更多信息，请参阅 AWS Resource Groups 入门。

以下模板的前两部分指定资源和资源组。模板的最后一部分为资源组创建 Application Insights 应用程序，但不配置应用程序或应用监控。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication 命令详细信息。

JSON 格式的模板

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Resource Group stack",
    "Resources": {
        "EC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId" : "ami-abcd1234efgh5678i",
                "SecurityGroupIds" : ["sg-abcd1234"]
            }
        },
        ...
        "ResourceGroup": {
            "Type": "AWS::ResourceGroups::Group",
            "Properties": {
                "Name": "my_resource_group"
            }
        },
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group"
            },
            "DependsOn" : "ResourceGroup"
        }
    }
}
```

YAML 格式的模板

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
```

```
      ImageId: ami-abcd1234efgh5678i
      SecurityGroupIds:
      - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
    DependsOn: ResourceGroup
```

以下模板部分将默认监控配置应用于 Application Insights 应用程序。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication 命令详细信息。

当 AutoConfigurationEnabled 设置为 true 时，将使用 DEFAULT 应用程序层的建议监控设置来配置应用程序的所有组件。有关这些设置和层的更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 DescribeComponentConfigurationRecommendation 和 UpdateComponentConfiguration。

JSON 格式的模板

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Application Insights Application stack",
    "Resources": {
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group",
                "AutoConfigurationEnabled": true
            }
        }
    }
}
```

YAML 格式的模板

```
---
AWSTemplateFormatVersion: '2010-09-09'
```

```
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

创建具有详细设置的 Application Insights 应用程序

以下模板执行以下操作：

- 创建启用了 CloudWatch Events 通知并启用 OpsCenter 的 Application Insights 应用程序。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication 命令详细信息。
- 使用两个标签来标记应用程序，其中一个标签没有标签值。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 TagResource。
- 创建两个自定义实例组组件。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateComponent。
- 创建两个日志模式集。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateLogPattern。
- 将 `AutoConfigurationEnabled` 设置为 `true`，这会使用 `DEFAULT` 层的建议监控设置来配置应用程序的所有组件。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 DescribeComponentConfigurationRecommendation。

JSON 格式的模板

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "CWEMonitorEnabled": true,
        "OpsCenterEnabled": true,
        "OpsItemSNSTopicArn": "arn:aws:sns:us-east-1:123456789012:my_topic",
        "AutoConfigurationEnabled": true,
        "Tags": [
            {
                "Key": "key1",
                "Value": "value1"
```

```
            },
            {
                "Key": "key2",
                "Value": ""
            }
        ],
        "CustomComponents": [
            {
                "ComponentName": "test_component_1",
                "ResourceList": [
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            },
            {
                "ComponentName": "test_component_2",
                "ResourceList": [
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i",
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            }
        ],
        "LogPatternSets": [
            {
                "PatternSetName": "pattern_set_1",
                "LogPatterns": [
                    {
                        "PatternName": "deadlock_pattern",
                        "Pattern": ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))",
                        "Rank": 1
                    }
                ]
            },
            {
                "PatternSetName": "pattern_set_2",
                "LogPatterns": [
                    {
                        "PatternName": "error_pattern",
                        "Pattern": ".*[\\s\\[]ERROR[\\s\\]].*",
                        "Rank": 1
                    },
                    {
                        "PatternName": "warning_pattern",
                        "Pattern": ".*[\\s\\[]WARN(ING)?[\\s\\]].*",
                        "Rank": 10
```

```
                    }
                ]
            }
        ]
    }
}
```

YAML 格式的模板

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  CWEMonitorEnabled: true
  OpsCenterEnabled: true
  OpsItemSNSTopicArn: arn:aws:sns:us-east-1:123456789012:my_topic
  AutoConfigurationEnabled: true
  Tags:
  - Key: key1
    Value: value1
  - Key: key2
    Value: ''
  CustomComponents:
  - ComponentName: test_component_1
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  - ComponentName: test_component_2
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  LogPatternSets:
  - PatternSetName: pattern_set_1
    LogPatterns:
    - PatternName: deadlock_pattern
      Pattern: ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))"
      Rank: 1
  - PatternSetName: pattern_set_2
    LogPatterns:
    - PatternName: error_pattern
      Pattern: ".*[\\s\\[]ERROR[\\s\\]].*"
      Rank: 1
    - PatternName: warning_pattern
      Pattern: ".*[\\s\\[]WARN(ING)?[\\s\\]].*"
```

```
      Rank: 10
```

使用 **CUSTOM** 模式组件配置创建 Application Insights 应用程序

以下模板执行以下操作：

- 创建 Application Insights 应用程序。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication。
- 组件 `my_component` 将 `ComponentConfigurationMode` 设置为 `CUSTOM`，这将导致按照 `CustomComponentConfiguration` 中指定的配置来配置此组件。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 UpdateComponentConfiguration。

JSON 格式的模板

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group,
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "CUSTOM",
                "CustomComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            },
                            ...
                        ],
                        "Logs": [
                            {
                                "LogGroupName": "my_log_group_1",
                                "LogPath": "C:\\LogFolder_1\\*",
                                "LogType": "DOT_NET_CORE",
                                "Encoding": "utf-8",
                                "PatternSet": "my_pattern_set_1"
                            },
                            ...
                        ],
```

```
                    "WindowsEvents": [
                        {
                            "LogGroupName": "my_windows_event_log_group_1",
                            "EventName": "Application",
                            "EventLevels": [
                                "ERROR",
                                "WARNING",
                                ...
                            ],
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_2"
                        },
                        ...
                    ],
                    "Alarms": [
                        {
                            "AlarmName": "my_alarm_name",
                            "Severity": "HIGH"
                        },
                        ...
                    ]
                },
                "SubComponentTypeConfigurations": [
                    {
                        "SubComponentType": "EC2_INSTANCE",
                        "SubComponentConfigurationDetails": {
                            "AlarmMetrics": [
                                {
                                    "AlarmMetricName": "DiskReadOps"
                                },
                                ...
                            ],
                            "Logs": [
                                {
                                    "LogGroupName": "my_log_group_2",
                                    "LogPath": "C:\\LogFolder_2\\*",
                                    "LogType": "IIS",
                                    "Encoding": "utf-8",
                                    "PatternSet": "my_pattern_set_3"
                                },
                                ...
                            ],
                            "processes" : [
                                {
```

```
                                    "processName" : "my_process",
                                    "alarmMetrics" : [
                                {
                                    "alarmMetricName" : "procstat cpu_usage",
                                    "monitor" : true
                                }, {
                                    "alarmMetricName" : "procstat memory_rss",
                                    "monitor" : true
                                }
                            ]
                        }
                    ],
                            "WindowsEvents": [
                                {
                                    "LogGroupName": "my_windows_event_log_group_2",
                                    "EventName": "Application",
                                    "EventLevels": [
                                        "ERROR",
                                        "WARNING",
                                        ...
                                    ],
                                    "Encoding": "utf-8",
                                    "PatternSet": "my_pattern_set_4"
                                },
                                ...
                            ]
                        }
                    }
                ]
            }
        }
    ]
}
}
```

YAML 格式的模板

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
```

```
    Tier: SQL_SERVER
    ComponentConfigurationMode: CUSTOM
    CustomComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
        ...
        Logs:
        - LogGroupName: my_log_group_1
          LogPath: C:\LogFolder_1\*
          LogType: DOT_NET_CORE
          Encoding: utf-8
          PatternSet: my_pattern_set_1
        ...
        WindowsEvents:
        - LogGroupName: my_windows_event_log_group_1
          EventName: Application
          EventLevels:
          - ERROR
          - WARNING
          ...
          Encoding: utf-8
          PatternSet: my_pattern_set_2
        ...
        Alarms:
        - AlarmName: my_alarm_name
          Severity: HIGH
        ...
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          AlarmMetrics:
          - AlarmMetricName: DiskReadOps
          ...
          Logs:
          - LogGroupName: my_log_group_2
            LogPath: C:\LogFolder_2\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set_3
          ...
          Processes:
          - ProcessName: my_process
            AlarmMetrics:
```

```
            - AlarmMetricName: procstat cpu_usage
              ...
            ...
          WindowsEvents:
          - LogGroupName: my_windows_event_log_group_2
            EventName: Application
            EventLevels:
            - ERROR
            - WARNING
            ...
            Encoding: utf-8
            PatternSet: my_pattern_set_4
          ...
```

使用 **DEFAULT** 模式组件配置创建 Application Insights 应用程序

以下模板执行以下操作：

- 创建 Application Insights 应用程序。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication。
- 组件 `my_component` 将 `ComponentConfigurationMode` 设置为 `DEFAULT`，并将 `Tier` 设置为 `SQL_SERVER`，这将导致使用 Application Insights 为 `SQL_Server` 层推荐的配置设置来配置此组件。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 DescribeComponentConfiguration 和 UpdateComponentConfiguration。

JSON 格式的模板

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "DEFAULT"
            }
        ]
    }
}
```

YAML 格式的模板

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: DEFAULT
```

使用 DEFAULT_WITH_OVERWRITE 模式组件配置创建 Application Insights 应用程序

以下模板执行以下操作：

- 创建 Application Insights 应用程序。有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 CreateApplication。
- 组件 my_component 将 ComponentConfigurationMode 设置为 DEFAULT_WITH_OVERWRITE，并将 tier 设置为 DOT_NET_CORE，这将导致使用 Application Insights 为 DOT_NET_CORE 层推荐的配置设置来配置此组件。覆盖的配置设置在 DefaultOverwriteComponentConfiguration 中指定：
  - 在组件级别，AlarmMetrics 设置将被覆盖。
  - 在子组件级别，对于 EC2_Instance 类型的子组件，将覆盖 Logs 设置。

  有关更多信息，请参阅 Amazon CloudWatch Application Insights API 参考中的 UpdateComponentConfiguration。

JSON 格式的模板

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentName": "my_component",
                "Tier": "DOT_NET_CORE",
                "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
                "DefaultOverwriteComponentConfiguration": {
                    "ConfigurationDetails": {
```

```
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            }
                        ]
                    },
                    "SubComponentTypeConfigurations": [
                        {
                            "SubComponentType": "EC2_INSTANCE",
                            "SubComponentConfigurationDetails": {
                                "Logs": [
                                    {
                                        "LogGroupName": "my_log_group",
                                        "LogPath": "C:\\LogFolder\\*",
                                        "LogType": "IIS",
                                        "Encoding": "utf-8",
                                        "PatternSet": "my_pattern_set"
                                    }
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

YAML 格式的模板

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentName: my_component
    Tier: DOT_NET_CORE
    ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
    DefaultOverwriteComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
      SubComponentTypeConfigurations:
```

```
    - SubComponentType: EC2_INSTANCE
      SubComponentConfigurationDetails:
        Logs:
        - LogGroupName: my_log_group
          LogPath: C:\LogFolder\*
          LogType: IIS
          Encoding: utf-8
          PatternSet: my_pattern_set
```

教程：为 SAP ASE 设置监控

本教程演示了如何配置 CloudWatch Application Insights 来为 SAP ASE 数据库设置监控。您可以使用 CloudWatch Application Insights 自动控制面板可视化问题详细信息、加快问题排查速度，并缩短 SAP ASE 数据库的平均解决时间（MTTR）。

Application Insights for SAP ASE 主题

- 支持的环境
- 支持的操作系统
- 功能
- 先决条件
- 设置针对 SAP ASE 数据库的监控
- 管理 SAP HANA 数据库的监控
- 配置告警阈值
- 查看和排查 Application Insights 检测到的 SAP ASE 问题
- 利用 Application Insights 排查 SAP ASE 的问题

支持的环境

CloudWatch Application Insights 支持部署用于以下系统和模式的 AWS 资源。您需提供并安装 SAP ASE 数据库软件和支持的 SAP 应用程序软件。

- 单个 Amazon EC2 实例上有一个或多个 SAP ASE 数据库 – 采用单节点、纵向扩展架构的 SAP ASE。
- 跨可用区 SAP ASE 数据库高可用性设置 – 使用 SUSE/RHEL 集群在两个可用区中配置了高可用性的 SAP ASE。

> **Note**
>
> CloudWatch Application Insights 仅支持单个 SAP 系统（SID）ASE HA 环境。如果附加了多个 ASE HA SID，则只会为检测到的第一个 SID 设置监控。

支持的操作系统

CloudWatch Application Insights for SAP ASE 在以下操作系统中支持 x86-64 架构：

- SuSE Linux 12 SP4
- SuSE Linux 12 SP5
- SUSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 SP3
- SuSE Linux 15 SP4
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- RedHat Linux 7.6
- RedHat Linux 7.7
- RedHat Linux 7.9
- RedHat Linux 8.1
- RedHat Linux 8.4
- RedHat Linux 8.6

功能

CloudWatch Application Insights for SAP ASE 具有以下功能：

- 自动检测 SAP ASE 工作负载
- 基于静态阈值自动创建 SAP ASE 警告
- 基于异常检测自动创建 SAP ASE 警告
- 自动识别 SAP ASE 日志模式
- SAP ASE 运行状况控制面板
- SAP ASE 问题控制面板

先决条件

您必须完成以下必需任务才能使用 CloudWatch Application Insights 配置 SAP ASE 数据库：

- SAP ASE 配置参数 – 必须在 ASE 数据库上启用以下配置参数："enable monitoring"、"sql text pipe max messages"、"sql text pipe active"。这让 CloudWatch Application Insights 可以为您的数据库提供全面的监控功能。如果您的 ASE 数据库未启用这些设置，Application Insights 将自动启用它们，以收集所需指标来进行监控。
- SAP ASE 数据库用户 – Application Insights 添加资源期间提供的数据库用户必须具有访问以下各项的权限：
  - 主数据库和用户（租户）数据库中的系统表
  - 监控表
- SAPHostCtrl – 在 Amazon EC2 实例上安装和设置 SAPHostCtrl。
- Amazon CloudWatch 代理 – 确保 Amazon EC2 实例上没有运行预先已有的 CloudWatch 代理。如果您安装了 CloudWatch 代理，请确保从现有 CloudWatch 代理配置文件中删除您在 CloudWatch Application Insights 中使用的资源的配置，以避免合并冲突。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。
- AWS Systems Manager 启用 – 在实例上安装 SSM Agent，并启用要为 SSM 启用的实例。有关如何安装 SSM Agent 的信息，请参阅 AWS Systems Manager 用户指南中的使用 SSM Agent。
- Amazon EC2 实例角色 – 要配置数据库，您必须附加以下 Amazon EC2 实例角色。
  - 要启用 Systems Manager，您必须附加 AmazonSSMManagedInstanceCore 角色。有关更多信息，请参阅 AWS Systems Manager 基于身份的策略示例。
  - 要启用通过 CloudWatch 发出实例指标和日志，您必须附加 CloudWatchAgentServerPolicy。有关更多信息，请参阅创建 IAM 角色和用户以用于 Amazon CloudWatch 代理。

- 要读取存储在 AWS Secrets Manager 中的密码，您必须将以下 IAM 内联策略附加到 Amazon EC2 实例角色。有关内联策略的更多信息，请参阅 AWS Identity and Access Management 用户指南中的内联策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- AWS Resource Groups – 要将应用程序添加到 CloudWatch Application Insights，您必须创建一个包含应用程序堆栈所用的所有相关 AWS 资源的资源组。这包括运行 SAP ASE 数据库的 Amazon EC2 实例和 Amazon EBS 卷。如果每个账户有多个数据库，我们建议您创建一个资源组，且该资源组包含每个 SAP ASE 数据库系统的 AWS 资源。
- IAM 权限 – 对于非管理员用户：
  - 您必须创建允许 Application Insights 创建服务相关角色的 AWS Identity and Access Management（IAM）policy，并将其附加到您的用户身份。有关附加策略的步骤，请参阅 IAM policy。
  - 用户必须有在 AWS Secrets Manager 中创建密钥的权限，以存储数据库用户凭证。有关更多信息，请参阅示例：创建密钥的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
```

```
}
```

- 服务相关角色 – Application Insights 使用 AWS Identity and Access Management（IAM）服务相关角色。当您在 Application Insights 控制台中创建首个 Application Insights 应用程序时，将会为您创建服务相关角色。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。

设置针对 SAP ASE 数据库的监控

使用以下步骤设置针对 SAP HANA 数据库的监控

1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。在右上角，选择 Add an application（添加应用程序）。
4. 在指定应用程序详细信息页面上，从资源组下的下拉列表中选择包含 SAP ASE 数据库资源的 AWS 资源组。如果尚未为应用程序创建资源组，则可以在 Resource group（资源组）下拉列表下选择 Create new resource group（创建新资源组）来创建一个资源组。有关创建资源组的更多信息，请参阅 AWS Resource Groups 用户指南。
5. 在 Monitor CloudWatch Events（监控 CloudWatch Events）下，选中复选框将 Application Insights 监控与 CloudWatch Events 集成，以获取 Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health API 和通知、Amazon RDS、Amazon S3 和 AWS Step Functions 中的洞察。
6. 在 Integrate with AWS Systems Manager OpsCenter（与 OpsCenter 集成）下，选中 Generate AWS Systems Manager OpsCenter OpsItems for remedial actions（生成 OpsCenter OpsItems 以采取修复措施）旁边的复选框，以在检测到所选应用程序的问题时查看问题及接收通知。要跟踪为解析与 AWS 资源相关的操作工作项 (OpsItem) 而执行的操作，请提供 SNS 主题 ARN。
7. 您可以选择输入标签以帮助您标识和整理资源。CloudWatch Application Insights 支持基于标签和基于 AWS CloudFormation 堆栈的资源组（Application Auto Scaling 组除外）。有关更多信息，请参阅 AWS Resource Groups 和标签用户指南中的标签编辑器。
8. 选择 Next（下一步）继续设置监控。
9. 查看检测到的组件页面上列出了 CloudWatch Application Insights 自动检测到的受监控组件及其工作负载。

> **Note**
>
> 包含检测到的 SAP ASE 高可用性工作负载的组件仅支持一个组件上一个工作负载。包含检测到的 SAP ASE 单节点工作负载的组件支持多个工作负载，但您无法添加或删除工作负载。所有自动检测到的工作负载都将受到监控。

10. 选择下一步。
11. 在指定组件详细信息页面上，输入 SAP ASE 数据库的用户名和密码。
12. 查看应用程序监控配置，然后选择 Submit（提交）。
13. 此时将打开应用程序详细信息页面，您可以在该页面中查看应用程序摘要、受监控组件和工作负载的列表以及未受监控的组件和工作负载。如果您选择组件或工作负载旁边的单选按钮，还可以查看配置历史记录、日志模式和已创建的任何标签。提交配置时，您的账户会为 SAP ASE 系统部署所有指标和警告，这最多可能需要 2 个小时。

管理 SAP HANA 数据库的监控

您可以执行以下步骤来管理 SAP ASE 数据库的用户凭证、指标和日志路径：

1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。
4. 在 Monitored components（已监控组件）下，选择组件名称旁边的单选按钮。然后，选择 Manage monitoring（管理监控）。
5. 在 EC2 instance group logs（EC2 实例组日志）下，您可以更新现有的日志路径、日志模式集和日志组名称。此外，您可以额外添加最多三个 Application logs（应用程序日志）。
6. 在指标下，您可以根据自己的要求选择 SAP ASE 指标。SAP ASE 指标名称的前缀是 `asedb`。每个组件可以添加最多 60 个指标。
7. 在 ASE 配置下，输入 SAP ASE 数据库的用户名和密码。即 Amazon CloudWatch 代理连接到 SAP ASE 数据库时使用的用户名和密码。
8. 在Custom alarms（自定义告警）下，您可以添加可由 CloudWatch Application Insights 监控的额外告警。
9. 查看应用程序监控配置并选择 Submit（提交）。提交配置时，您的账户会为 SAP HANA 系统更新所有指标和告警，这最多可能需要 2 个小时。

配置告警阈值

CloudWatch Application Insights 会自动创建由告警监控的 Amazon CloudWatch 指标，以及该指标的阈值。当该指标在指定数量的评估期内超出阈值，告警将变为 ALARM （告警）状态。请注意，Application Insights 不会保留这些设置。

要编辑单个指标的告警，请按以下步骤操作：

1. 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，依次选择 Alarms（告警）和 All alarms（所有告警）。
3. 选择 CloudWatch Application Insights 自动创建的告警旁边的单选按钮。然后选择 Actions（操作），并选择下拉菜单中的 Edit（编辑）。
4. 编辑 Metric（指标）下的以下参数。
   a. 在 Statistic（统计数据）下，选择其中一个统计数据或预定义百分比值，或指定一个自定义百分比值。例如，p95.45。
   b. 在 Period（时间段）下，选择告警的评估期。评估告警时，每个时间段都聚合到一个数据点。
5. 编辑 Conditions（条件）下的以下参数。
   a. 选择指标是否必须大于、小于或等于阈值。
   b. 指定阈值。
6. 在 Additional configuration（其他配置）下，编辑以下参数：
   a. 在 Datapoints to alarm（触发告警的数据点数）下，指定必须处于 ALARM（告警）状态以启动告警的数据点数或评估期。当两个值匹配时，如果超过指定的连续评估期数，将创建一个告警并进入 ALARM（告警）状态。要创建“m（最大为 n）”告警，则为第一个数据点指定的值应小于为第二个数据点指定的值。有关评估告警的更多信息，请参阅评估告警。
   b. 在 Missing data treatment（缺失数据处理）下，选择在缺失某些数据点时的告警行为。有关缺失数据处理的更多信息，请参阅配置 CloudWatch 告警处理缺失数据的方式。
   c. 如果警报将百分比值作为监控的统计数据，将显示样本数少的百分比框。选择是评估还是忽略采样率较低的案例。如果选择忽略 (保持警报状态)，在样本大小太小时，将始终保持当前警报状态。有关采样较低的百分位数的更多信息，请参阅 基于百分位数的 CloudWatch 告警和小数据样本。
7. 选择下一步。

8. 在通知下面，选择一个在警报处于 ALARM、OK 或 INSUFFICIENT_DATA 状态时通知的 SNS 主题。
9. 选择 Update alarm（更新告警）。

查看和排查 Application Insights 检测到的 SAP ASE 问题

本节可帮助您解决在 Application Insights 上针对 SAP ASE 配置监控时出现的常见问题。

SAP ASE Backup 服务器错误

您可以通过检查动态创建的控制面板来识别错误消息。控制面板会显示 SAP ASE Backup 服务器中报告的错误消息。有关 SAP ASE Backup Server 日志的更多详细信息，请参阅 SAP 文档“Backup Server 错误记录”。

SAP ASE 长时间运行的事务

确定长时间运行的事务，并确认是否可以将其停止或运行时间是否有意设置的。有关更多详细信息，请参阅 2180410 – How to display transaction log records for long running transactions? – SAP ASE。

SAP ASE 用户连接

检查您的 SAP ASE 数据库的大小是否与您打算在数据库上运行的工作负载相匹配。有关更多详细信息，请参阅 SAP 文档中的 Configuring User Connections。

SAP ASE 磁盘空间

您可以检查动态创建的控制面板，找出导致问题的数据库层。控制面板会显示相关指标和日志文件片段。请务必了解磁盘扩容的原因，并在适用情况下增加物理磁盘大小、分配的磁盘空间或同时增加两者。有关更多详细信息，请参阅 SAP 文档中的 disk resize。

利用 Application Insights 排查 SAP ASE 的问题

本节提供了帮助您解决 Application Insights 控制面板返回的常见错误的步骤。

错误	返回的错误	根本原因	解决方案
无法添加超过 60 个监控指标。	`Component cannot have more than`	当前的指标限制为每个组件 60 个受监控指标。	删除不必要的指标以符合限制。

错误	返回的错误	根本原因	解决方案
	`60 monitored metric`		
添加过程之后未显示 SAP 指标或警告	`run` 上的 `AWS-ConfigureAWSPackage` 命令在 AWS Systems Manager 中失败了。输出显示错误：`CT-LIBRARY error:ct_connect(): protocol specific layer: external error: The attempt to connect to the server failed`	用户名和密码可能不正确。	验证用户名和密码是否有效，然后重新运行添加过程。

教程：为 SAP HANA 设置监控

本教程演示了如何配置 CloudWatch Application Insights 来为 SAP HANA 数据库设置监控。您可以使用 CloudWatch Application Insights 自动控制面板可视化呈现问题详细信息、加快故障排除速度及缩短 SAP HANA 数据库的平均解决时间 (MTTR)。

Application Insights for SAP HANA 主题

- 支持的环境
- 支持的操作系统
- 功能
- 先决条件
- 设置 SAP HANA 数据库以进行监控
- 管理对 SAP HANA 数据库的监控
- 查看和排查 CloudWatch Application Insights 检测到的问题
- SAP HANA 的异常检测
- 排查 Application Insights for SAP HANA 的问题

支持的环境

CloudWatch Application Insights 支持部署用于以下系统和模式的 AWS 资源。您提供并安装 SAP HANA 数据库软件和支持的 SAP 应用程序软件。

- 单个 Amazon EC2 实例上的 SAP HANA 数据库 – 采用单节点、纵向扩展架构的 SAP HANA，内存高达 24TB。
- 多个 Amazon EC2 实例上的 SAP HANA 数据库 – 采用多节点、横向扩展架构的 SAP HANA。
- 跨可用区 SAP HANA 数据库高可用性设置 – 使用 SUSE/RHEL 集群在两个可用区中配置了高可用性的 SAP HANA。

> **Note**
>
> CloudWatch Application Insights 仅支持单个 SID HANA 环境。如果连接了多个 HANA SID，则只会为第一个检测到的 SID 设置监控。

支持的操作系统

CloudWatch Application Insights for SAP HANA 在以下操作系统中支持 x86-64 架构：

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- SUSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 15 SP5 For SAP
- 适用于 SAP 的 RedHat Linux 8.6（包含高可用性和更新服务）
- 适用于 SAP 的 RedHat Linux 8.5（包含高可用性和更新服务）

- 适用于 SAP 的 RedHat Linux 8.4（包含高可用性和更新服务）
- 适用于 SAP 的 RedHat Linux 8.3（包含高可用性和更新服务）
- RedHat Linux 8.2 For SAP（包含高可用性和更新服务）
- RedHat Linux 8.1 For SAP（包含高可用性和更新服务）
- RedHat Linux 7.9 For SAP（包含高可用性和更新服务）

功能

CloudWatch Application Insights for SAP HANA 具有以下功能：

- SAP HANA 工作负载自动检测
- 基于静态阈值自动创建 SAP HANA 告警
- 基于异常检测自动创建 SAP HANA 告警
- 自动识别 SAP HANA 日志模式
- SAP HANA 健康控制面板
- SAP HANA 问题控制面板

先决条件

您必须完成以下必需任务才能使用 CloudWatch Application Insights 配置 SAP HANA 数据库：

- SAP HANA – 在 Amazon EC2 实例上安装处于运行状态且可访问的 SAP HANA 数据库 2.0 SPS05。
- SAP HANA 数据库用户 – 必须在 SYSTEM 数据库和所有租户中创建具有监控角色的数据库用户。

  示例

  以下 SQL 命令将创建具有监控角色的用户。

```
su - <sid>adm
hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
 FORCE_FIRST_PASSWORD_CHANGE;
CREATE ROLE CW_HANADB_EXPORTER_ROLE;
GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
```

- Python 3.8 – 在操作系统上安装 Python 3.8 或更高版本。使用最新的 Python 版本。如果未在操作系统中检测到 Python3，则将安装 Python 3.6。

  有关更多信息，请参阅 installation example。

  > **Note**
  >
  > SuSE Linux 15 SP4、RedHat Linux 8.6 及更高版本的操作系统需要手动安装 Python 3.8 或更高版本。

- Pip3 – 在操作系统中安装安装程序 pip3。如果未在操作系统中检测到 pip3，则安装该程序。
- hdbclient – CloudWatch Application Insights 使用 python 驱动连接到 SAP HANA 数据库。如果未在 python3 下安装客户端，则请确保 `/hana/shared/SID/hdbclient/` 下有 hdbclient tar 文件版本 `2.10 or later`。
- Amazon CloudWatch 代理 – 确保 Amazon EC2 实例上没有运行预先已有的 CloudWatch 代理。如果您安装了 CloudWatch 代理，请确保从现有 CloudWatch 代理配置文件中删除您在 CloudWatch Application Insights 中使用的资源的配置，以避免合并冲突。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。
- AWS Systems Manager 启用 – 在实例上安装 SSM Agent，并且实例必须启用了 SSM。有关如何安装 SSM Agent 的信息，请参阅《AWS Systems Manager 用户指南》中的使用 SSM Agent。
- Amazon EC2 实例角色 – 要配置数据库，您必须附加以下 Amazon EC2 实例角色。
  - 要启用 Systems Manager，您必须附加 `AmazonSSMManagedInstanceCore` 角色。有关更多信息，请参阅 AWS Systems Manager 基于身份的策略示例。
  - 要启用通过 CloudWatch 发出实例指标和日志，您必须附加 `CloudWatchAgentServerPolicy`。有关更多信息，请参阅创建用于 CloudWatch 代理的 IAM 角色和用户
  - 要读取存储在 AWS Secrets Manager 中的密码，您必须将以下 IAM 内联策略附加到 Amazon EC2 实例角色。有关内联策略的更多信息，请参阅 AWS Identity and Access Management 用户指南中的内联策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
```

```
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- AWS Resource Groups – 要将应用程序添加到 CloudWatch Application Insights，您必须创建一个包含应用程序堆栈所用的所有相关 AWS 资源的资源组。这包括运行 SAP HANA 数据库的 Amazon EC2 实例和 Amazon EBS 卷。如果每个账户有多个数据库，我们建议您创建一个资源组，而且该资源组包含每个 SAP HANA 数据库系统的 AWS 资源。
- IAM 权限 – 对于非管理员用户：
  - 您必须创建允许 Application Insights 创建服务相关角色的 AWS Identity and Access Management（IAM）policy，并将其附加到您的用户身份。有关附加策略的步骤，请参阅 IAM policy。
  - 用户必须有在 AWS Secrets Manager 中创建密钥的权限，以存储数据库用户凭证。有关更多信息，请参阅示例：创建密钥的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- 服务相关角色 – Application Insights 使用 AWS Identity and Access Management（IAM）服务相关角色。当您在 Application Insights 控制台中创建首个 Application Insights 应用程序时，将会为您创建服务相关角色。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。

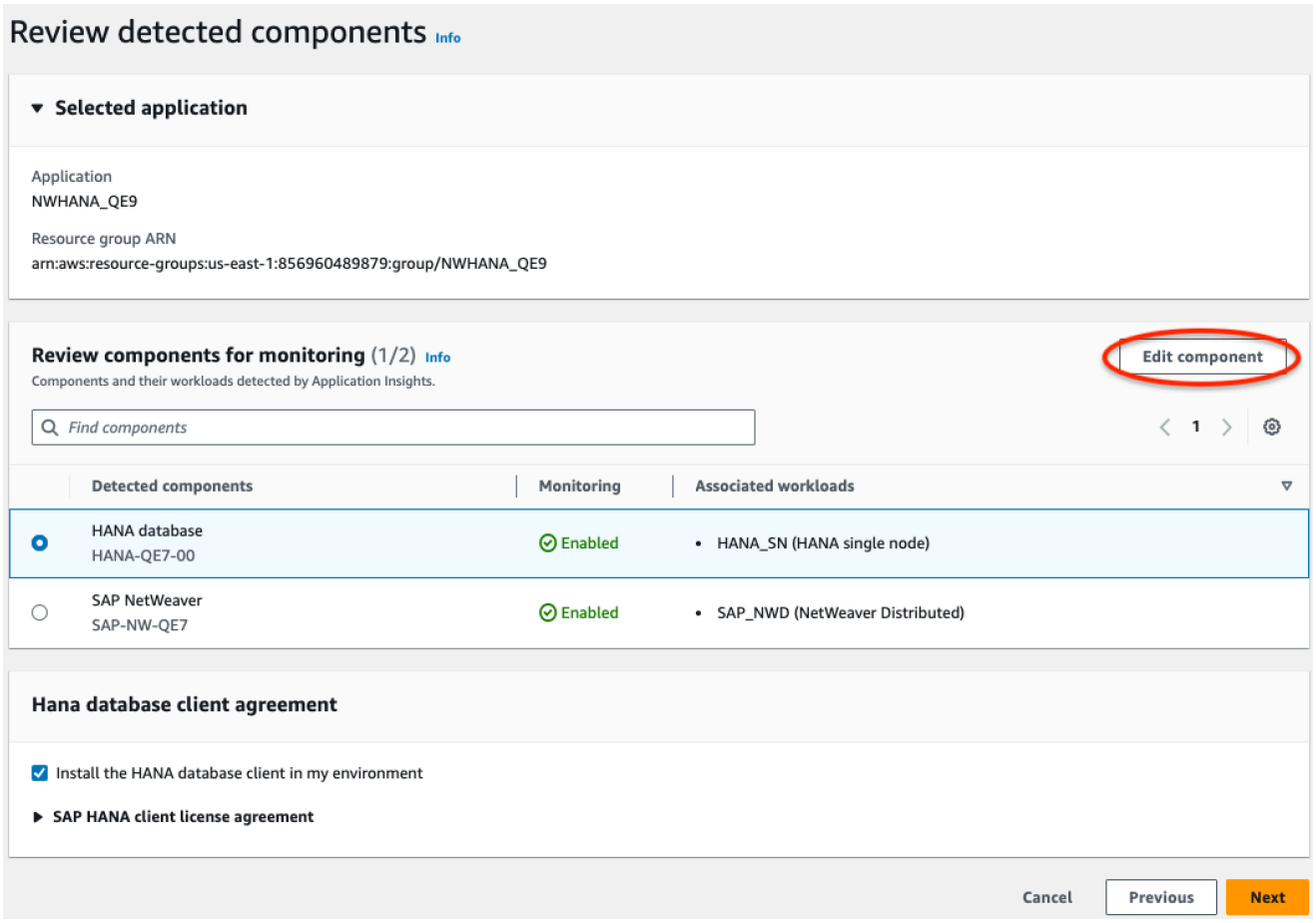
设置 SAP HANA 数据库以进行监控

要为 SAP HANA 数据库设置监控，请执行以下步骤

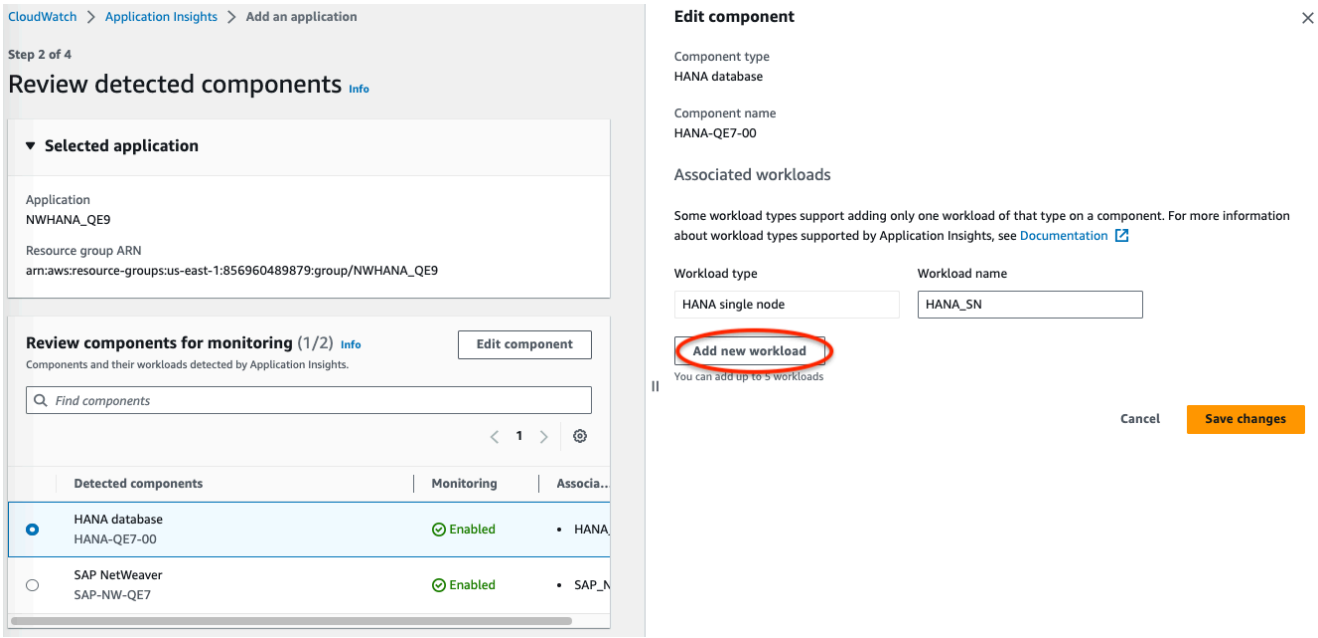
1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。在右上角，选择 Add an application（添加应用程序）。
4. 在 Specify application details（指定应用程序详细信息）页面上，从 Resource group（资源组）下的下拉列表中选择包含 SAP HANA 数据库资源的 AWS 资源组。如果尚未为应用程序创建资源组，则可以在 Resource group（资源组）下拉列表下选择 Create new resource group（创建新资源组）来创建一个资源组。有关创建资源组的更多信息，请参阅 AWS Resource Groups 用户指南。
5. 在 Monitor CloudWatch Events（监控 CloudWatch Events）下，选中复选框将 Application Insights 监控与 CloudWatch Events 集成，以获取 Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health API 和通知、Amazon RDS、Amazon S3 和 AWS Step Functions 中的洞察。
6. 在 Integrate with AWS Systems Manager OpsCenter（与 OpsCenter 集成）下，选中 Generate AWS Systems Manager OpsCenter OpsItems for remedial actions（生成 OpsCenter OpsItems 以采取修复措施）旁边的复选框，以在检测到所选应用程序的问题时查看问题及接收通知。要跟踪为解析与 AWS 资源相关的操作工作项 (OpsItem) 而执行的操作，请提供 SNS 主题 ARN。
7. 您可以选择输入标签以帮助您标识和整理资源。CloudWatch Application Insights 支持基于标签和基于 AWS CloudFormation 堆栈的资源组（Application Auto Scaling 组除外）。有关更多信息，请参阅 AWS Resource Groups 和标签用户指南中的标签编辑器。
8. 选择 Next（下一步）继续设置监控。
9. 查看检测到的组件页面上列出了 CloudWatch Application Insights 自动检测到的受监控组件及其工作负载。

    a. 要将工作负载添加到包含检测到的 SAP HANA 单节点工作负载的组件，请选择该组件，然后选择编辑组件。

> **Note**
>
> 包含检测到的 SAP HANA 多节点或 HANA 高可用性工作负载的组件仅支持一个组件上一个工作负载。



b. 要添加新的工作负载，请选择添加新工作负载。



c. 编辑工作负载完成后，选择保存更改。

10. 选择下一步。
11. 在指定组件详细信息页面上，输入用户名和密码。
12. 查看应用程序监控配置，然后选择 Submit（提交）。
13. 此时将打开应用程序详细信息页面，您可以在该页面中查看应用程序摘要、受监控组件和工作负载的列表以及未受监控的组件和工作负载。如果您选择组件或工作负载旁边的单选按钮，还可以查看配置历史记录、日志模式和已创建的任何标签。提交配置时，您的账户会为 SAP HANA 系统部署所有指标和告警，这最多可能需要 2 个小时。

管理对 SAP HANA 数据库的监控

您可以执行以下步骤来管理 SAP HANA 数据库的用户凭证、指标和日志路径：

1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。
4. 在 Monitored components（已监控组件）下，选择组件名称旁边的单选按钮。然后，选择 Manage monitoring（管理监控）。
5. 在 EC2 instance group logs（EC2 实例组日志）下，您可以更新现有的日志路径、日志模式集和日志组名称。此外，您可以额外添加最多三个 Application logs（应用程序日志）。
6. 在 Metrics（指标）下，您可以根据自己的要求选择 SAP HANA 指标。SAP HANA 指标名称的前缀是 hanadb。每个组件可以添加最多 40 个指标。
7. 在 HANA configuration（HANA 配置）下，输入 SAP HANA 数据库的密码和用户名，即 Amazon CloudWatch 代理连接到 SAP HANA 数据库时使用的用户名和密码。
8. 在Custom alarms（自定义告警）下，您可以添加可由 CloudWatch Application Insights 监控的额外告警。
9. 查看应用程序监控配置并选择 Submit（提交）。提交配置时，您的账户会为 SAP HANA 系统更新所有指标和告警，这最多可能需要 2 个小时。

查看和排查 CloudWatch Application Insights 检测到的问题

以下各节提供了一些步骤，可帮助您解决在 Application Insights 上为 SAP HANA 配置监控时出现的常见问题排查情况。

故障排除主题

- SAP HANA 数据库达到内存分配限制
- 磁盘已满事件
- SAP HANA 备份停止运行

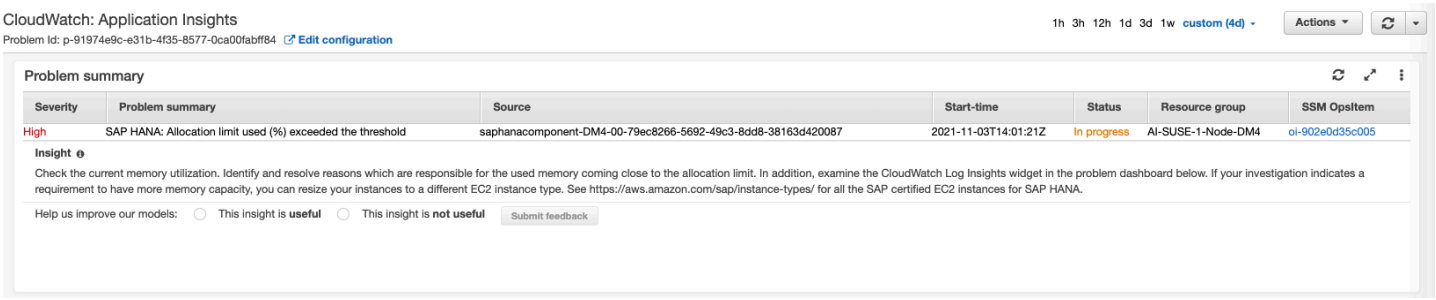
SAP HANA 数据库达到内存分配限制

描述

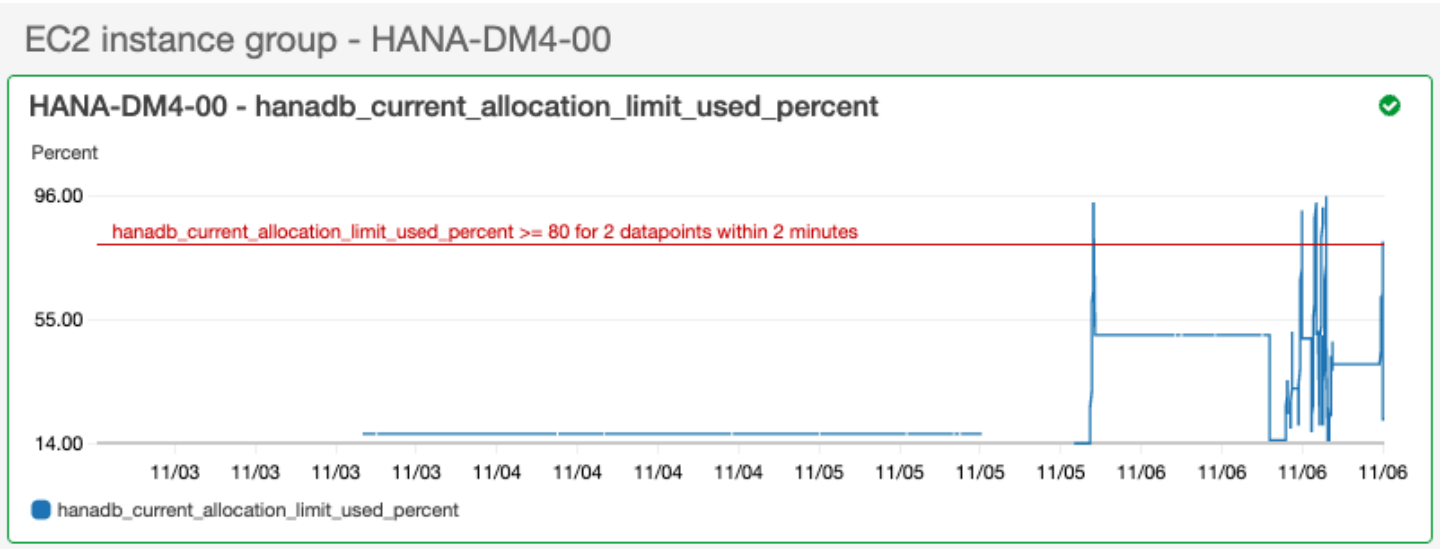
由于内存压力过大，由 SAP HANA 数据库提供支持的 SAP 应用程序发生故障，导致应用程序性能下降。

解决方案

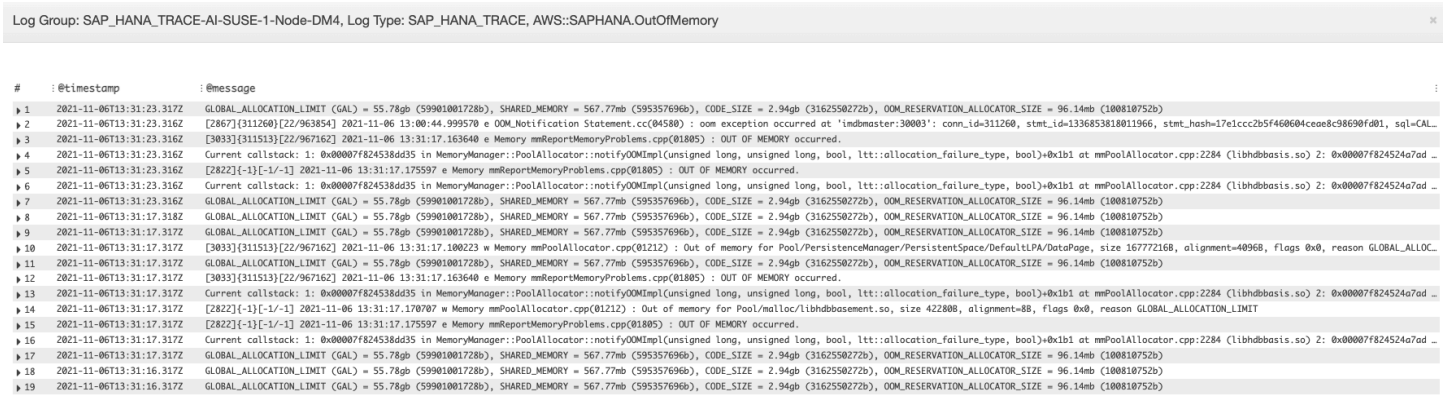
您可以检查动态创建的控制面板（显示相关指标和日志文件片段），找出导致问题的应用程序层。在以下示例中，问题可能是 SAP HANA 系统中数据负载过大。



已用内存分配超过总内存分配限制的 80% 的阈值。



日志组显示方案 BNR-DATA 和表 IMDBMASTER_30003 内存不足。此外，日志组会显示问题发生的确切时间、当前全局位置限制、共享内存、代码大小和 OOM 预留分配大小。



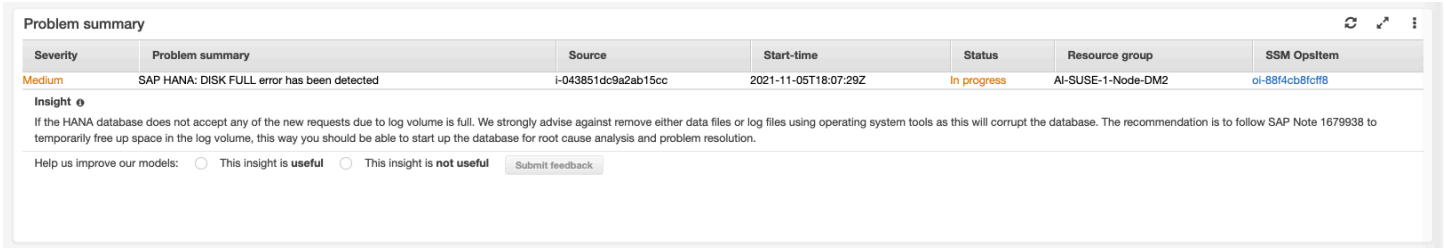
磁盘已满事件

描述

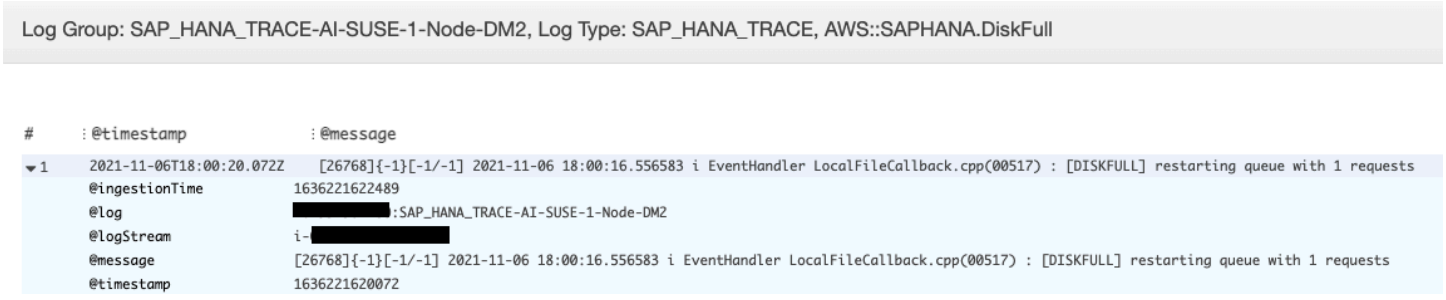
由 SAP HANA 数据库提供支持的 SAP 应用程序停止响应，导致无法访问数据库。

解决方案

您可以检查动态创建的控制面板（显示相关指标和日志文件片段），找出导致问题的数据库层。在以下示例中，问题可能是管理员无法启用自动日志备份，导致 sap/hana/log 目录填满。



问题控制面板中的日志组小组件显示 DISKFULL 事件。



SAP HANA 备份停止运行

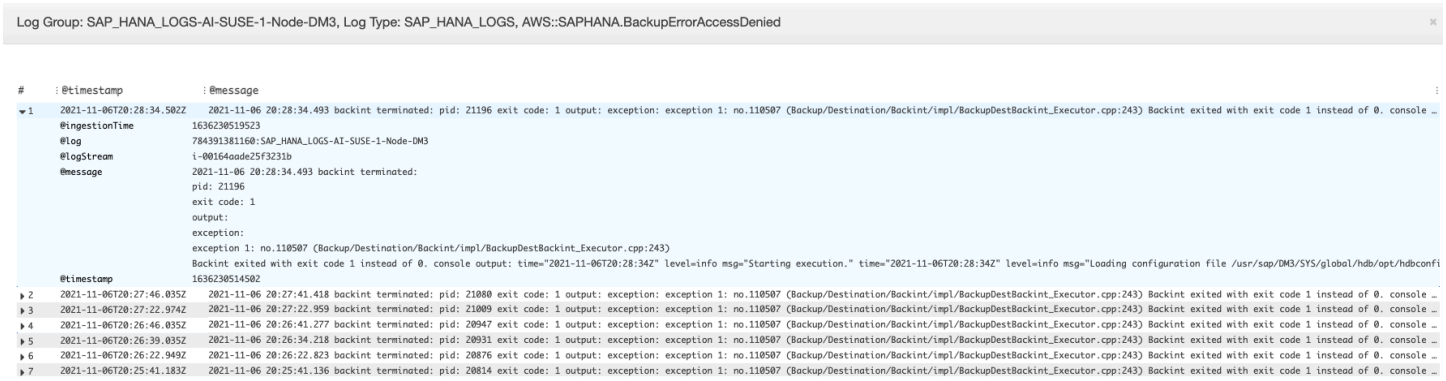
描述

由 SAP HANA 数据库提供支持的 SAP 应用程序已停止工作。

解决方案

您可以检查动态创建的控制面板（显示相关指标和日志文件片段），找出导致问题的数据库层。

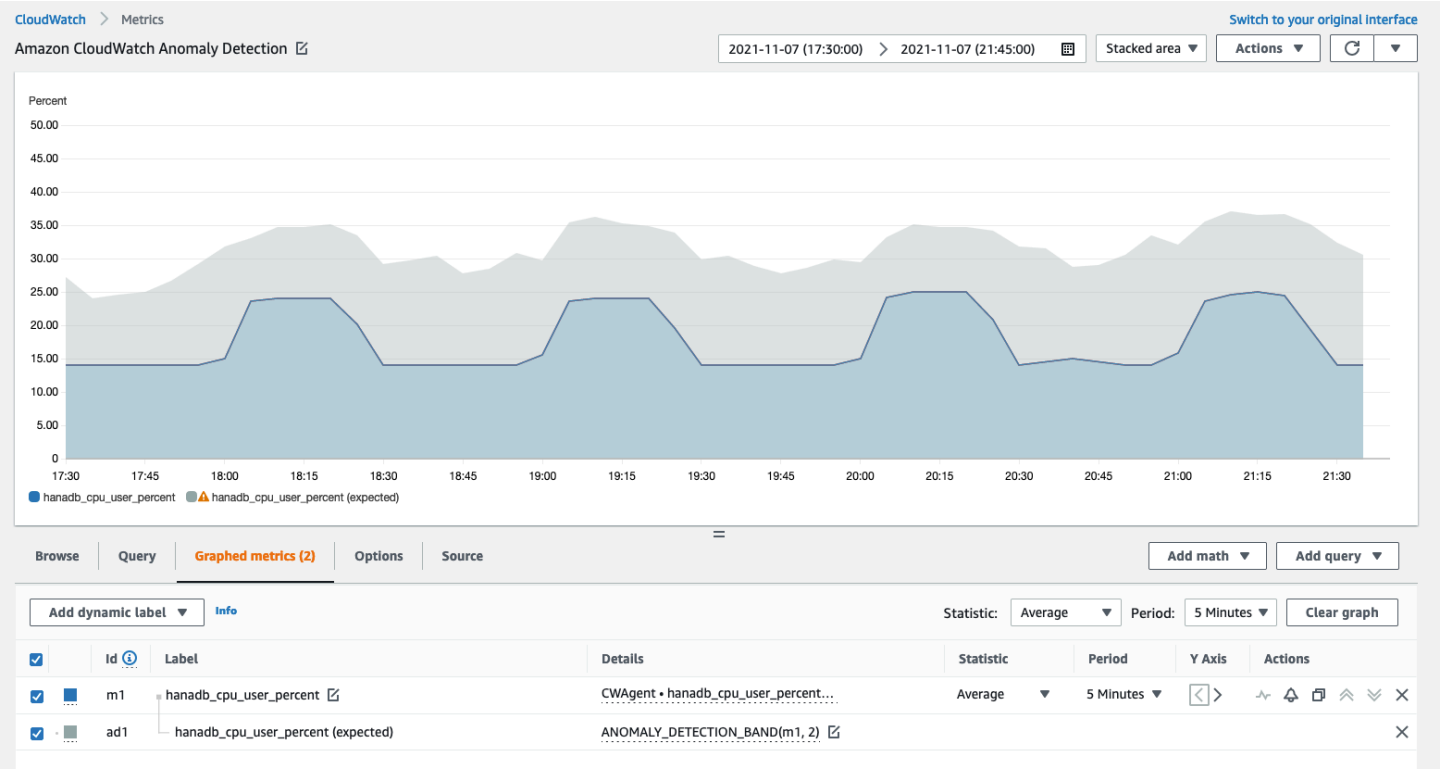
问题控制面板中的日志组小组件显示 ACCESS DENIED 事件。其中包括其他信息，如 S3 存储桶、S3 存储桶文件夹和 S3 存储桶区域。



SAP HANA 的异常检测

对于特定的 SAP HANA 指标（例如线程数量），CloudWatch 应用统计和机器学习算法来定义阈值。这些算法只需最少的用户干预，即可持续分析 SAP HANA 数据库的指标，确定正常基线和表面异常。算法会生成一个异常检测模型，该模型生成表示正常指标行为的预期值范围。

异常检测算法将指标的季节性变化和趋势变化考虑在内。季节性变化可以是每小时、每天或每周，如以下 SAP HANA CPU 使用率示例所示。



创建模型后，CloudWatch 异常检测会持续评估模型并对其进行调整，以确保模型尽可能准确。这包括重新训练模型，以便模型在指标值随着时间的推移而变化或突然发生变化时作出调整。此外还包括用于改进季节性、峰值或稀疏指标模型的预测器。

排查 Application Insights for SAP HANA 的问题

本节提供了帮助您解决 Application Insights 控制面板返回的常见错误的步骤。

无法添加超过 60 个监控的指标

输出内容显示以下错误。

```
Component cannot have more than 60 monitored metrics
```

根本原因 – 当前的指标限制为每个组件 60 个受监控指标。

解决方法 – 要保持在限制以下，请删除不必要的指标。

载入过程之后未显示 SAP 指标

使用以下信息了解 SAP 指标在载入过程后未显示在控制面板上的原因。第一步是使用 AWS Management Console 或 Amazon EC2 实例的导出程序日志来排查 SAP 指标不显示的原因。接下来，查看错误输出以找到解决方法。

排查载入后 SAP 指标不显示的原因

您可以使用 AWS Management Console 或 Amazon EC2 实例的导出程序日志进行问题排查。

AWS Management Console

使用控制台排查载入后不显示 SAP 指标的问题

1. 访问 https://console.aws.amazon.com/systems-manager/，打开 AWS Systems Manager 控制台。
2. 在左侧导航窗格中，选择状态管理器。
3. 在关联下，检查文档 `AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure` 的状态。如果状态为 `Failed`，则在执行 ID 下，选择失败的 ID 并查看输出。
4. 在关联下，检查文档 `AWS-ConfigureAWSPackage` 的状态。如果状态为 `Failed`，则在执行 ID 下，选择失败的 ID 并查看输出。

Exporter logs from Amazon EC2 instance

使用导出程序日志排查载入后不显示 SAP 指标的问题

1. 连接到运行 SAP HANA 数据库的 Amazon EC2 实例。
2. 使用以下命令查找用于 `WORKLOAD_SHORT_NAME` 的正确命名约定。您将在以下两个步骤中使用此短名称。

```
sudo systemctl | grep exporter
```

> **Note**
>
> Application Insights 根据正在运行的工作负载在服务名称中添加后缀 `WORKLOAD_SHORT_NAME`。SAP HANA 单节点、多节点和高可用性部署的短名称为 `HANA_SN`、`HANA_MN` 和 `HANA_HA`。

3. 要检查导出器管理器服务日志中是否存在错误，请运行以下命令，将 `WORKLOAD_SHORT_NAME` 替换为您在 Step 2 中找到的短名称。

```
sudo journalctl -e --unit=prometheus-
hanadb_exporter_manager_WORKLOAD_SHORT_NAME.service
```

4. 如果导出程序管理器服务日志未显示错误，则请运行以下命令检查导出程序服务日志中是否存在错误。

```
sudo journalctl -e --unit=prometheus-hanadb_exporter_WORKLOAD_SHORT_NAME.service
```

解决载入后 SAP 指标未显示的常见根本原因

以下示例描述了如何解决载入后 SAP 指标未显示的常见根本原因。

- 输出内容显示以下错误。

```
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-
cloudwatch-agent.d/default ...
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/
amazon-cloudwatch-agent.d/ssm_AmazonCloudWatch-ApplicationInsights-
SSMParameterForTESTCWEC2INSTANCEi0d88867f1f3e36285.tmp ...
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Under path : /metrics/append_dimensions | Error : Different
 values are specified for append_dimensions
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/disk | Error : Different
 values are specified for disk
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/mem | Error : Different
 values are specified for mem
2023/11/30 22:25:17 Configuration validation first phase failed. Agent version: 1.0.
 Verify the JSON input is only using features supported by this version.
```

解决方案 – Application Insights 尝试配置与现有 CloudWatch 代理配置文件中预配置指标相同的指标。删除 `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/` 下现有的文件，或从现有 CloudWatch 代理配置文件中删除导致冲突的指标。

- 输出内容显示以下错误。

```
Unable to find a host with system database, for more info rerun using -v
```

解决方案 – 用户名、密码或数据库端口可能不正确。验证用户名、密码和端口是否有效，然后重新运行载入过程。

- 输出内容显示以下错误。

```
This hdbcli installer is not compatible with your Python interpreter
```

分辨率 – 升级 pip3 和 wheel，如以下 Python 3.6 示例所示。

```
python3.6 -m pip install --upgrade pip setuptools wheel
```

- 输出内容显示以下错误。

```
Unable to install hdbcli using pip3. Please try to install it
```

解决方案 – 确保您已遵循 hdbclient 先决条件或在 pip3 下手动安装 hdbclient。

- 输出内容显示以下错误。

```
Package 'boto3' requires a different Python: 3.6.15 not in '>= 3.7'
```

解决方案 – 此操作系统版本需要 Python 3.8 或更高版本。检查 Python 3.8 的先决条件并进行安装。

- 输出内容显示以下安装错误之一。

```
Can not execute `setup.py` since setuptools is not available in the build environment
```

或者

```
[SSL: CERTIFICATE_VERIFY_FAILED]
```

解决方法 – 使用 SUSE Linux 命令安装 Python，如以下示例所示。以下示例安装了最新版本的 Python 3.8。

```
wget https://www.python.org/ftp/python/3.8.<LATEST_RELEASE>/
Python-3.8.<LATEST_RELEASE>.tgz
tar xf Python-3.*
cd Python-3.*/
sudo zypper install make gcc-c++ gcc automake autoconf libtool
sudo zypper install zlib-devel
```

```
sudo zypper install libopenssl-devel libffi-devel
./configure --with-ensurepip=install
sudo make
sudo make install
sudo su
python3.8 -m pip install --upgrade pip setuptools wheel
```

教程：为 SAP NetWeaver 设置监控

本教程演示了如何配置 Amazon CloudWatch Application Insights，以便为 SAP NetWeaver 设置监控。您可以使用 CloudWatch Application Insights 自动控制面板可视化问题详细信息、加快故障排除并缩短 SAP NetWeaver 应用程序服务器的平均解决时间（MTTR）。

CloudWatch Application Insights for SAP NetWeaver 主题

- 支持的环境
- 支持的操作系统
- 功能
- 先决条件
- 设置 SAP NetWeaver 应用程序服务器以进行监控
- 管理 SAP NetWeaver 应用程序服务器的监控
- 查看和排查 CloudWatch Application Insights 检测到的 SAP NetWeaver 问题
- 排查 Application Insights for SAP NetWeaver 的问题

支持的环境

CloudWatch Application Insights 支持部署用于以下系统和模式的 AWS 资源。

- SAP NetWeaver 标准系统部署。
- SAP NetWeaver 在多个 Amazon EC2 实例上的分布式部署。
- 跨可用区 SAP NetWeaver 高可用性设置 – 使用 SUSE/RHEL 集群在两个可用区之间配置高可用性的 SAP NetWeaver。

支持的操作系统

以下操作系统支持 CloudWatch Application Insights for SAP NetWeaver：

- Oracle Linux 8
- Red Hat Enterprise Linux 7.6
- Red Hat Enterprise Linux 7.7
- Red Hat Enterprise Linux 7.9
- Red Hat Enterprise Linux 8.1
- Red Hat Enterprise Linux 8.2
- Red Hat Enterprise Linux 8.4
- Red Hat Enterprise Linux 8.6
- SUSE Linux Enterprise Server 15 for SAP
- SUSE Linux Enterprise Server 15 SP1 for SAP
- SUSE Linux Enterprise Server 15 SP2 for SAP
- SUSE Linux Enterprise Server 15 SP3 for SAP
- SUSE Linux Enterprise Server 15 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP5 for SAP
- SUSE Linux Enterprise Server 15（高可用性模式除外）
- SUSE Linux Enterprise Server 15 SP1（高可用性模式除外）
- SUSE Linux Enterprise Server 15 SP2（高可用性模式除外）
- SUSE Linux Enterprise Server 15 SP3（高可用性模式除外）
- SUSE Linux Enterprise Server 15 SP4（高可用性模式除外）
- SUSE Linux Enterprise Server 12 SP4（高可用性模式除外）
- SUSE Linux Enterprise Server 12 SP5（高可用性模式除外）

功能

CloudWatch Application Insights for SAP NetWeaver 7.0x–7.5x（包括 ABAP 平台）提供了以下功能：

- SAP NetWeaver 工作负载自动检测
- 基于静态阈值自动创建 SAP NetWeaver 告警
- 自动识别 SAP NetWeaver 日志模式
- SAP NetWeaver 运行状况控制面板
- SAP NetWeaver 问题控制面板

先决条件

您须完成以下必需任务才能使用 CloudWatch Application Insights 配置 SAP NetWeaver：

- AWS Systems Manager 启用 – 在 Amazon EC2 实例上安装 SSM Agent，并启用 SSM 实例。有关如何安装 SSM Agent 的信息，请参阅 AWS Systems Manager 用户指南中的设置 AWS Systems Manager。
- Amazon EC2 实例角色 – 要配置 SAP NetWeaver 监控，您必须附加以下 Amazon EC2 实例角色。
  - 要启用 Systems Manager，您必须附加 `AmazonSSMManagedInstanceCore` 角色。有关更多信息，请参阅 AWS Systems Manager 基于身份的策略示例。
  - 要使实例指标和日志通过 CloudWatch 发出，您必须附加 `CloudWatchAgentServerPolicy` 策略。有关更多信息，请参阅创建用于 CloudWatch 代理的 IAM 角色和用户
- AWS Resource Groups – 要将应用程序添加到 CloudWatch Application Insights，您必须创建一个包含应用程序堆栈所用的所有相关 AWS 资源的资源组。这包括运行 SAP NetWeaver 应用程序服务器的 Amazon EC2 实例、Amazon EFS 和 Amazon EBS 卷。如果每个账户有多个 SAP NetWeaver 系统，我们建议您创建一个资源组，且该资源组包含每个 SAP NetWeaver 系统的 AWS 资源。有关创建资源组的更多信息，请参阅 AWS 资源组和标签用户指南。
- IAM 权限 – 对于没有管理访问权限的用户，必须创建允许 Application Insights 创建服务相关角色的 AWS Identity and Access Management（IAM）policy，并将其附加到用户身份。有关如何创建 IAM policy 的更多信息，请参阅 IAM policy。
- 服务相关角色 – Application Insights 使用 AWS Identity and Access Management（IAM）服务相关角色。当您在 Application Insights 控制台中创建首个 Application Insights 应用程序时，将会为您创建服务相关角色。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。
- Amazon CloudWatch 代理 – Application Insights 安装和配置 CloudWatch 代理。如果您安装了 CloudWatch 代理，则 Application Insights 会保留您的配置。为避免合并冲突，请从现有 CloudWatch 代理配置文件中删除您想在 Application Insights 中使用的资源的配置。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

设置 SAP NetWeaver 应用程序服务器以进行监控

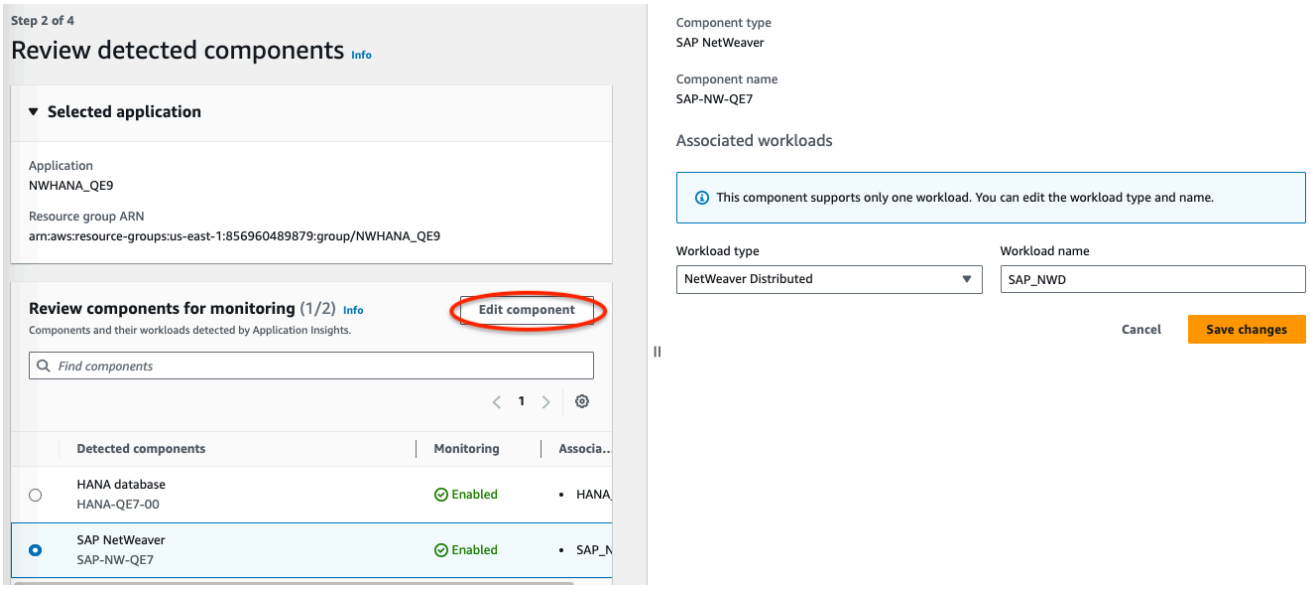
请按照以下步骤为 SAP NetWeaver 应用程序服务器设置监控。

设置监控

1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。在右上角，选择 Add an application（添加应用程序）。
4. 在 Specify application details（指定应用程序详细信息）页面上，从 Resource group（资源组）下拉列表中选择您创建的包含 SAP NetWeaver 资源的 AWS 资源组。如果尚未为应用程序创建资源组，则可以在 Resource group（资源组）下拉列表下选择 Create new resource group（创建新资源组）来创建一个资源组。
5. 在 Automatic monitoring of new resources（自动监控新资源）下，选中复选框以允许 Application Insights 自动监控在载入后添加到应用程序资源组的资源。
6. 在 Monitor EventBridge events（监控 EventBridge 事件）下，选中复选框将 Application Insights 监控与 CloudWatch Events 集成，以获取 Amazon EBS、Amazon EC2、AWS CodeDeploy、Amazon ECS、AWS Health API 和通知、Amazon RDS、Amazon S3 和 AWS Step Functions 中的洞察。
7. 在 Integrate with AWS Systems Manager OpsCenter（与 OpsCenter 集成）下，选中 Generate AWS Systems Manager OpsCenter OpsItems for remedial actions（生成 OpsCenter OpsItems 以采取修复措施）旁边的复选框，以在检测到所选应用程序的问题时查看问题及接收通知。要跟踪为解析与 AWS 资源相关的操作工作项（称为 OpsItems）而执行的操作，请提供 SNS 主题 ARN。
8. 您可以选择输入标签以帮助您标识和整理资源。CloudWatch Application Insights 支持基于标签和基于 AWS CloudFormation 堆栈的资源组（Application Auto Scaling 组除外）。有关更多信息，请参阅 AWS Resource Groups 和标签用户指南中的标签编辑器。
9. 要查看检测到的组件，请选择下一步。
10. 查看检测到的组件页面上列出了 CloudWatch Application Insights 自动检测到的受监控组件及其工作负载。

    - 如需编辑工作负载类型和名称，请选择编辑组件。

> **Note**
>
> 包含检测到的 NetWeaver 分布式或 NetWeaver 高可用性工作负载的组件仅支持一个组件一个工作负载。



11. 选择下一步。
12. 在 Specify component details（指定组件详细信息）页面上，选择 Next（下一步）。
13. 查看应用程序监控配置，然后选择提交。
14. 应用程序详细信息页面会打开，您可以在其中查看应用程序摘要、控制面板、组件和工作负载。您还可以查看 Configuration history（配置历史记录）、Log patterns（日志模式）和已创建的任何 Tags（标签）。在您提交应用程序后，CloudWatch Application Insights 会为您的 SAP NetWeaver 系统部署所有指标和告警，这最多可能需要一小时。

管理 SAP NetWeaver 应用程序服务器的监控

请按照以下步骤管理 SAP NetWeaver 应用程序服务器的监控。

管理监控

1. 打开 CloudWatch 控制台。
2. 从左侧导航窗格中，选择 Insights 下的 Application Insights。
3. 选择 List view（列表视图）选项卡。

4. Application Insights 页面会显示使用 Application Insights 监控的应用程序列表以及每个应用程序的监控状态。
5. 选择您的应用程序。
6. 选择 Components（组件）选项卡。
7. 在 Monitored components（已监控组件）下，选择组件名称旁边的单选按钮。然后，选择 Manage monitoring（管理监控）。
8. 在 Instance logs（实例日志）下，您可以更新现有的日志路径、日志模式集和日志组名称。此外，您可以额外添加最多三个 Application logs（应用程序日志）。
9. 在 Metrics（指标）下，您可以根据自己的要求选择 SAP NetWeaver 指标。SAP NetWeaver 指标名称的前缀是 `sap`。每个组件可以添加最多 40 个指标。
10. 在Custom alarms（自定义告警）下，您可以添加可由 CloudWatch Application Insights 监控的额外告警。
11. 查看应用程序监控配置并选择 Save（保存）。提交配置时，您的账户会为 SAP NetWeaver 系统更新所有指标和告警。

查看和排查 CloudWatch Application Insights 检测到的 SAP NetWeaver 问题

以下部分提供的步骤可帮助您解决在 Application Insights 上为 SAP NetWeaver 配置监控时出现的常见问题排查情况。

故障排除主题

- SAP NetWeaver 数据库连接问题
- SAP NetWeaver 应用程序可用性问题

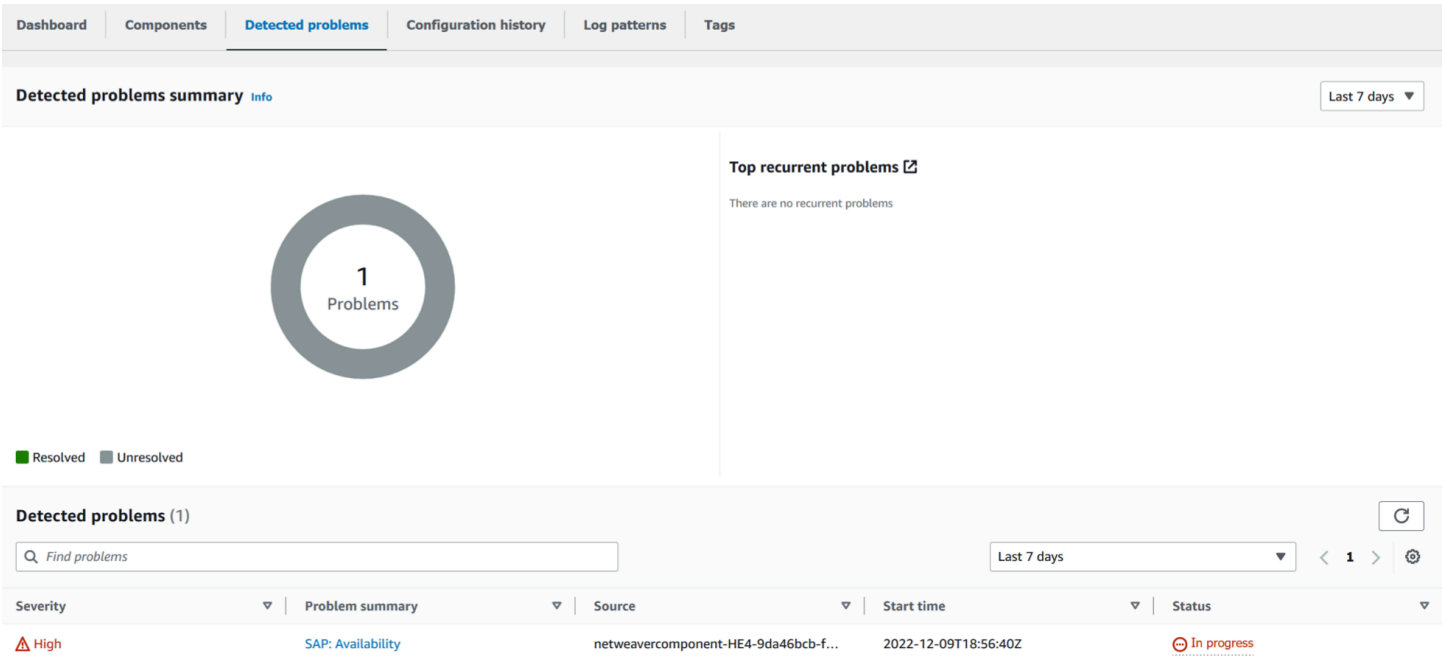
SAP NetWeaver 数据库连接问题

描述

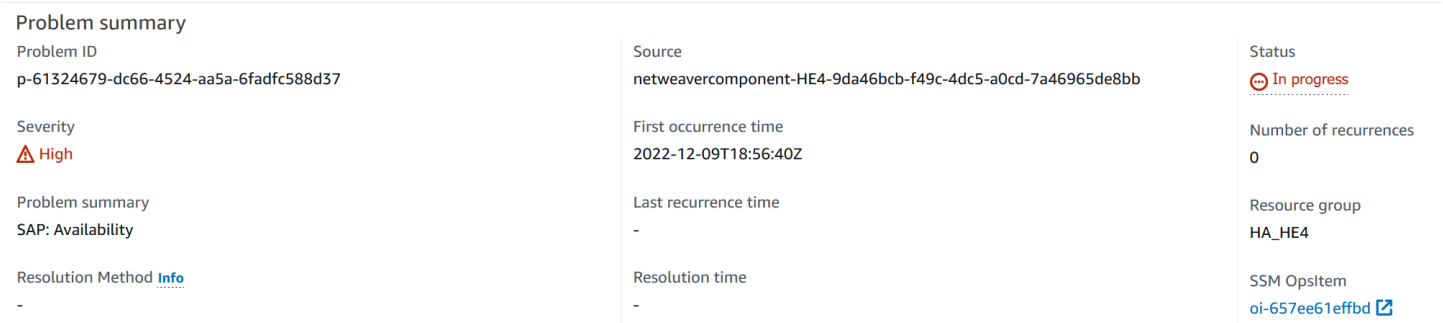
您的 SAP NetWeaver 应用程序出现数据库连接问题。

原因

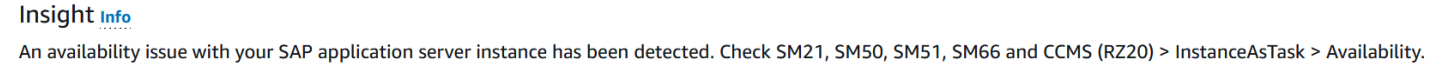
您可以前往 CloudWatch Application Insights 控制台并查看 SAP NetWeaver Application Insights 问题控制面板来确定连接问题。选择 Problem summary（问题摘要）下的链接以查看特定问题。



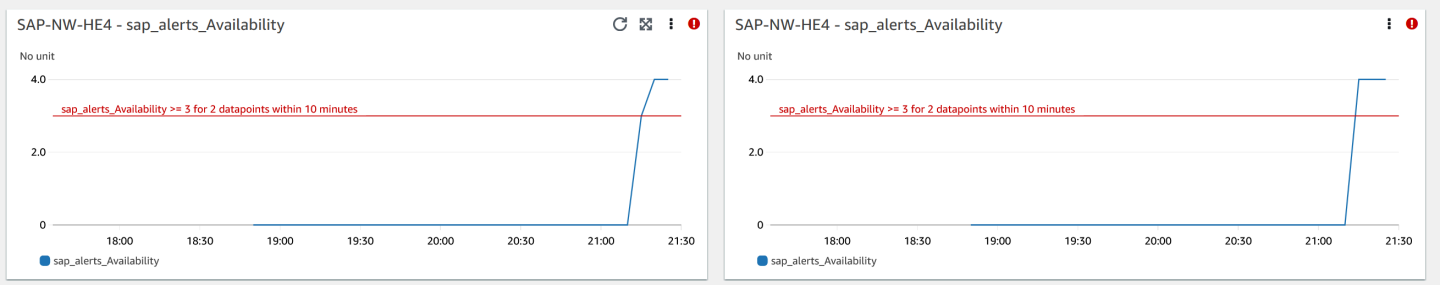
在以下示例中，Problem summary（问题摘要）下的 SAP: Availability（SAP：可用性）是出现的问题。



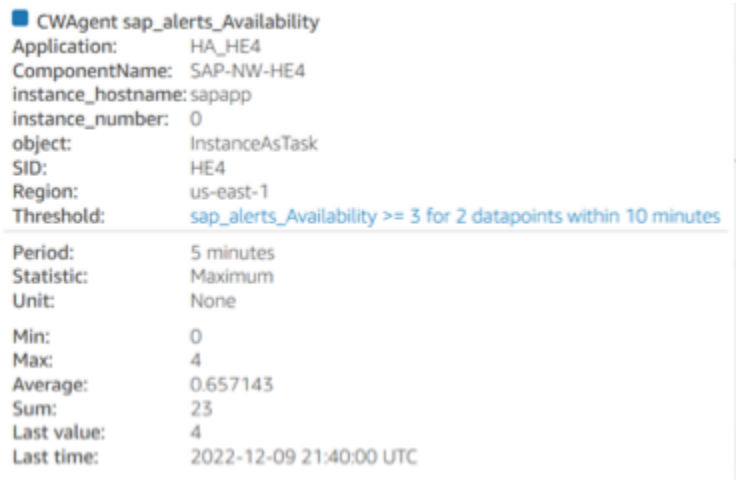
紧接在 Problem summary（问题摘要）之后，Insight（洞察）部分提供了有关错误的更多上下文，您可以从中获得有关问题原因的更多信息。



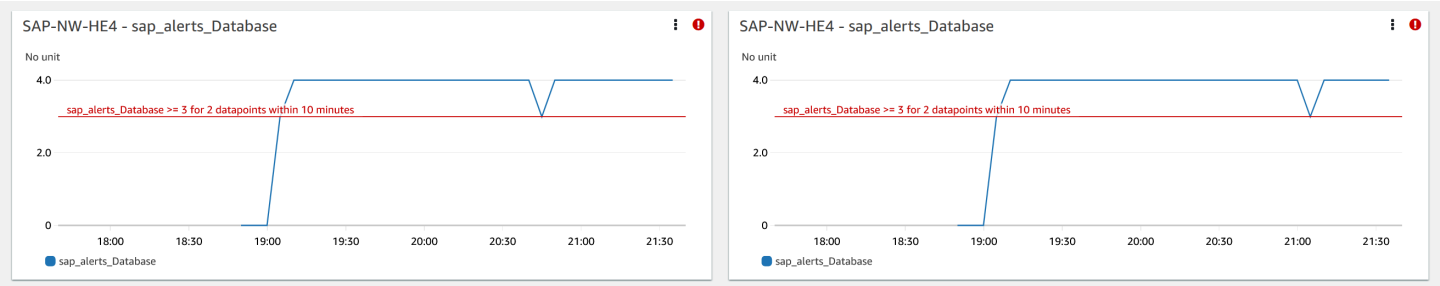
在同一个问题控制面板上，您可以查看问题检测功能汇总在一起的相关日志和指标，以帮助您找出错误原因。`sap_alerts_Availability` 指标可跟踪 SAP NetWeaver 系统随时间变化的可用性。您可以使用历史跟踪功能来关联该指标何时启动错误状态或超出告警阈值。在以下示例中，SAP NetWeaver 系统存在可用性问题。该示例显示了两个告警，因为有两个 SAP 应用程序服务器实例，系统为每个实例创建了一个告警。



如要了解有关每个告警的更多信息，请将鼠标悬停在 `sap_alerts_Availability` 指标名称上。



在以下示例中，`sap_alerts_Database` 指标显示数据库层存在问题或故障。该告警表明 SAP NetWeaver 在连接到其数据库或与其数据库通信时出现了问题。



由于数据库是 SAP NetWeaver 的关键资源，因此当数据库出现问题或故障时，您可能会收到许多相关的告警。在以下示例中，由于数据库不可用，系统启动了 `sap_alerts_FrontendResponseTime` 和 `sap_alerts_LongRunners` 指标。

解决方案

Application Insights 按小时监测检测到的问题。如果 SAP NetWeaver 日志文件中没有新的相关日志条目，则较旧的日志条目将被视为已解决。您必须修复与 CloudWatch 告警相关的任何错误条件。修复错误条件后，在恢复告警和日志时告警会得到解决。解决所有 CloudWatch 日志错误和告警后，Application Insights 将停止检测错误，问题将在一小时内自动解决。我们建议您解决所有日志错误条件和告警，这样您就可以在问题控制面板上看到最新的问题。

以下示例中解决了 SAP 可用性问题。

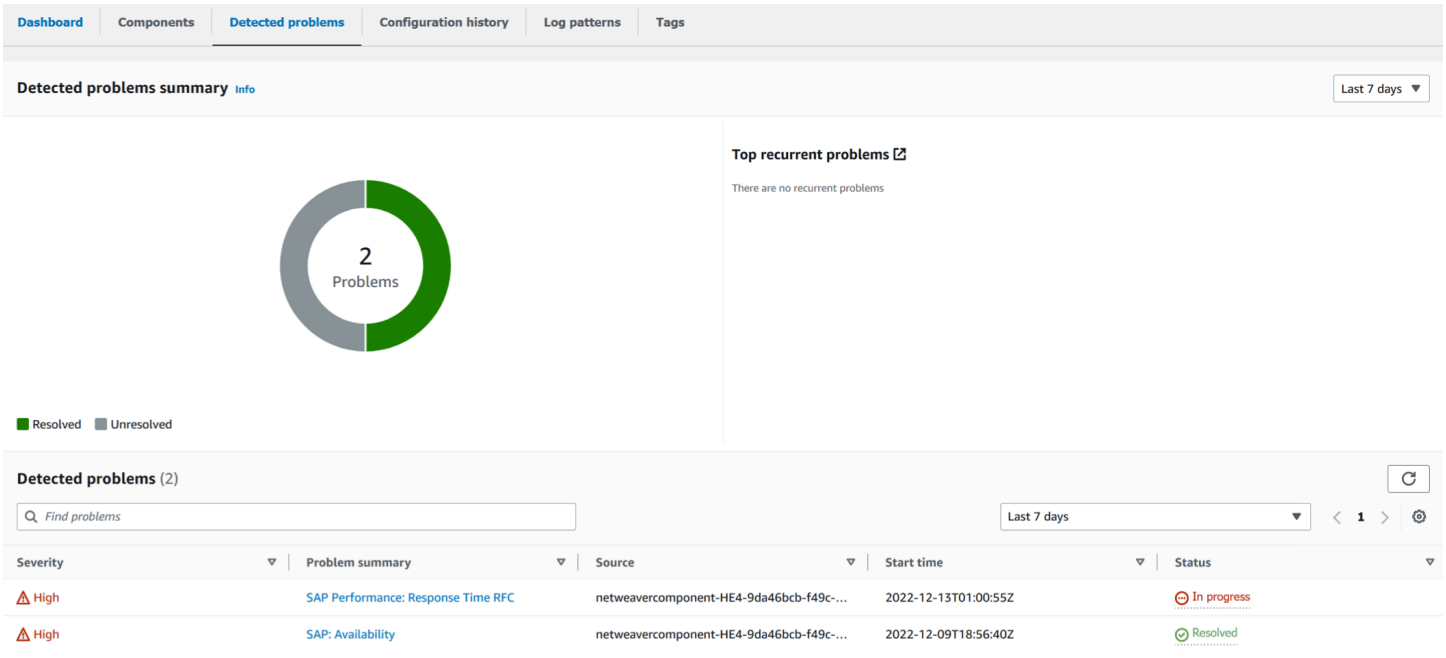
SAP NetWeaver 应用程序可用性问题

描述

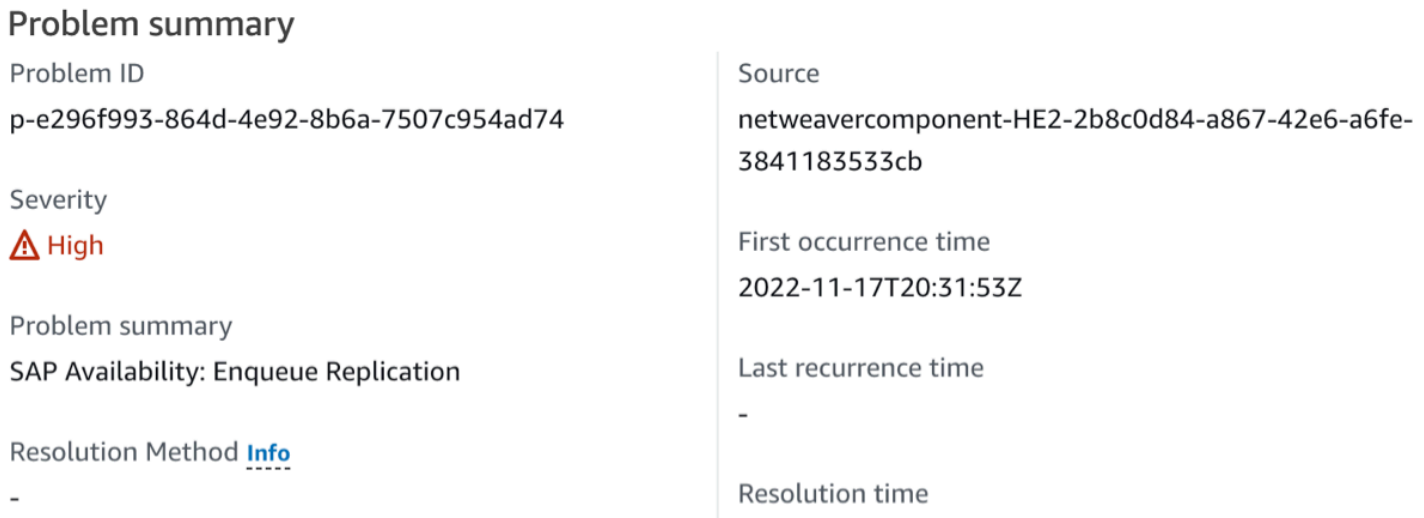
SAP NetWeaver 高可用性入队复制停止运行。

原因

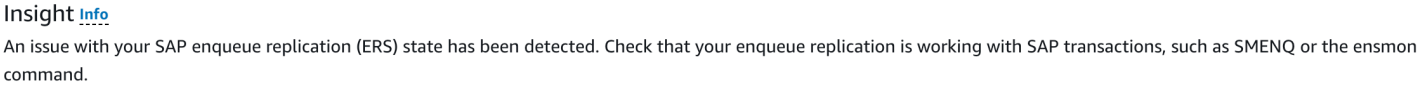
您可以前往 CloudWatch Application Insights 控制台并查看 SAP NetWeaver Application Insights 问题控制面板来确定连接问题。选择 Problem summary（问题摘要）下的链接以查看特定问题。



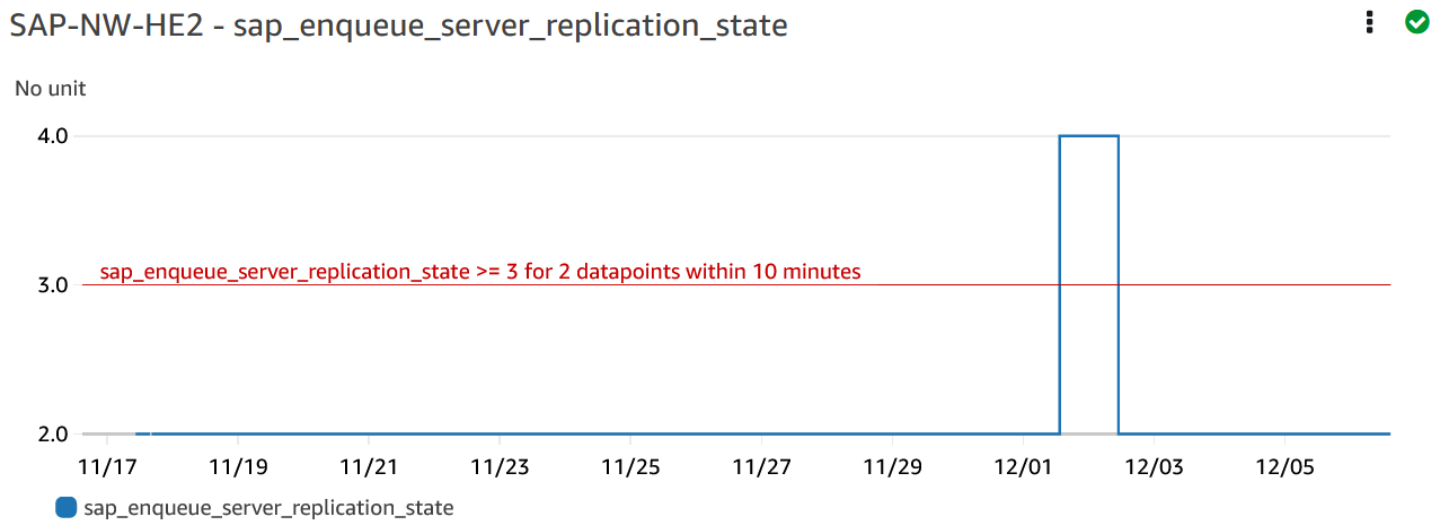
在以下示例中，Problem summary（问题摘要）下的高可用性入队复制是出现的问题。



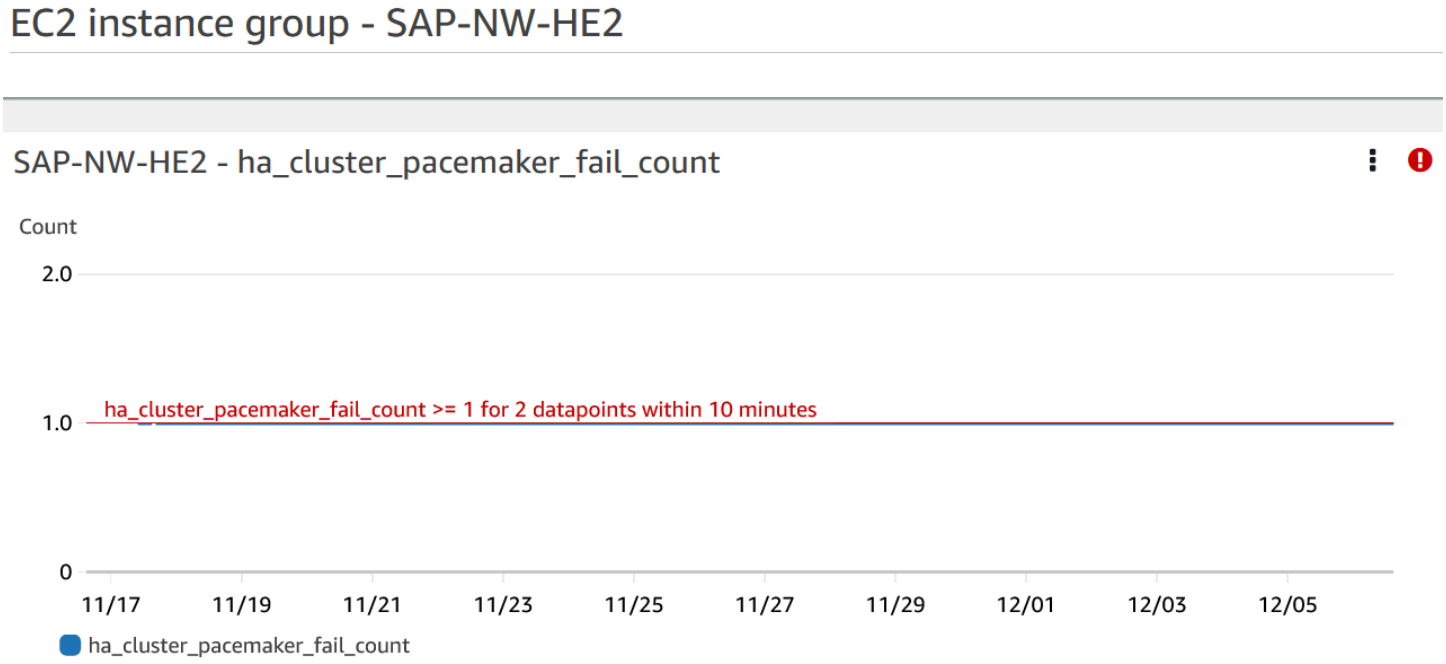
紧接在 Problem summary（问题摘要）之后，Insight（洞察）部分提供了有关错误的更多上下文，您可以从中获得有关问题原因的更多信息。



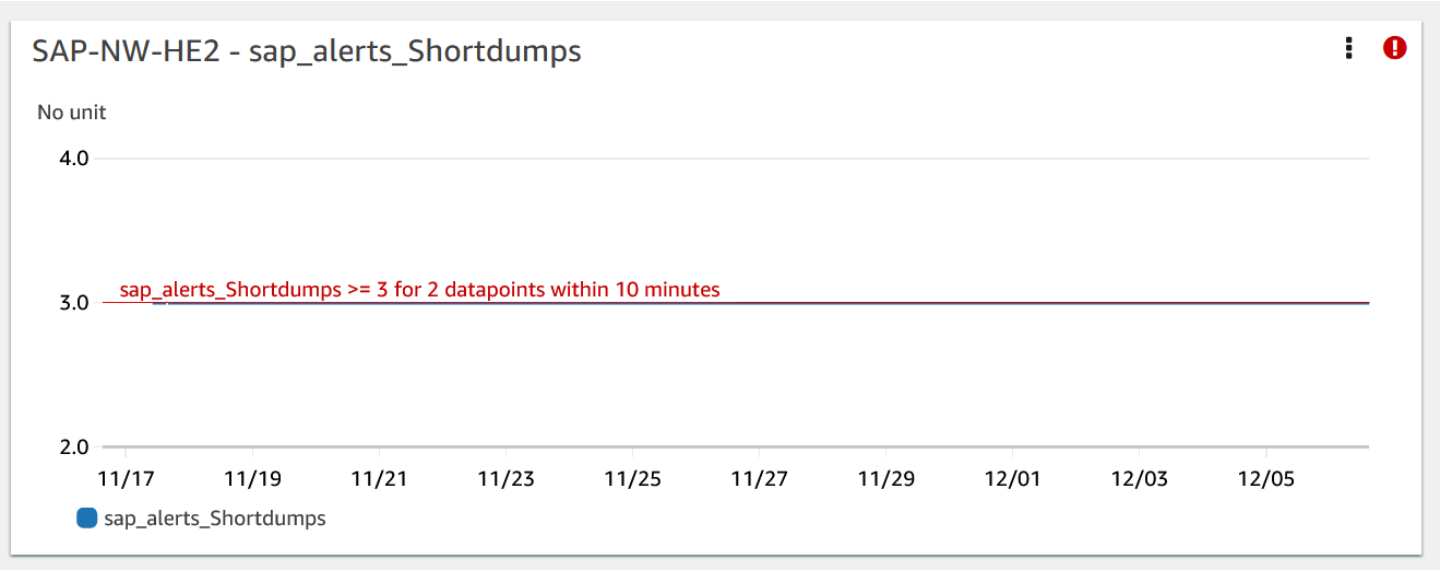
以下示例显示了问题控制面板，您可以在其中查看分组的日志和指标，以帮助您找出错误原因。`sap_enqueue_server_replication_state` 指标可随时间跟踪该值。您可以使用历史跟踪功能来关联该指标何时启动错误状态或超出告警阈值。



在以下示例中，ha_cluster_pacemaker_fail_count 指标显示高可用性 Pacemaker 集群出现了资源故障。组件控制面板中标识了故障次数大于或等于 1 的特定 Pacemaker 资源。

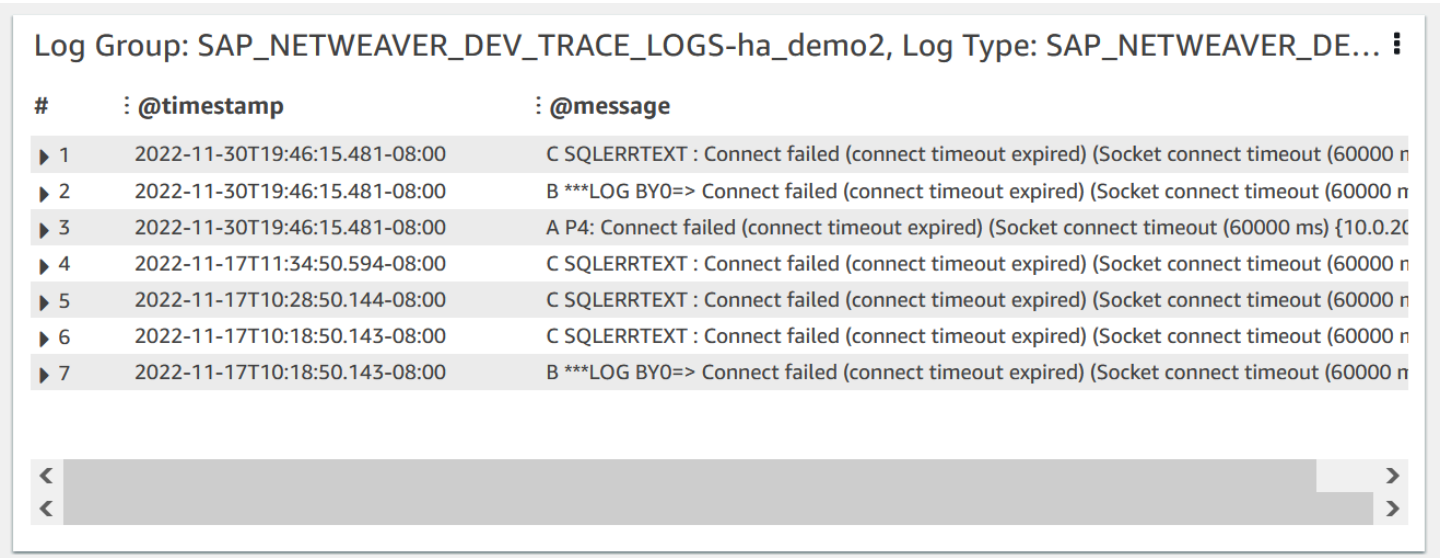


以下示例显示了 sap_alerts_Shortdumps 指标，该指标表明在检测到问题时 SAP 应用程序的性能有所降低。

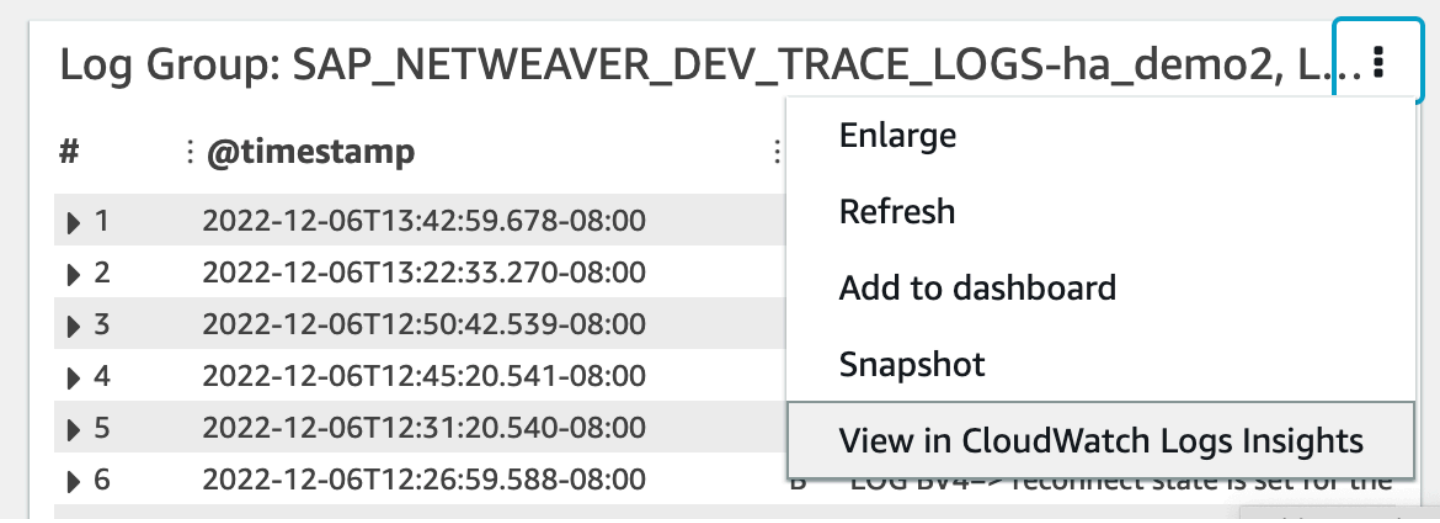


日志

这些日志条目有助于更好地了解在检测到问题时 SAP NetWeaver 层出现的问题。问题控制面板中的日志组小组件显示了问题发生的具体时间。



要查看有关日志的详细信息，请选择右上角的三个垂直点，然后选择 View in CloudWatch Logs Insights（在 CloudWatch Logs Insights 中查看）。

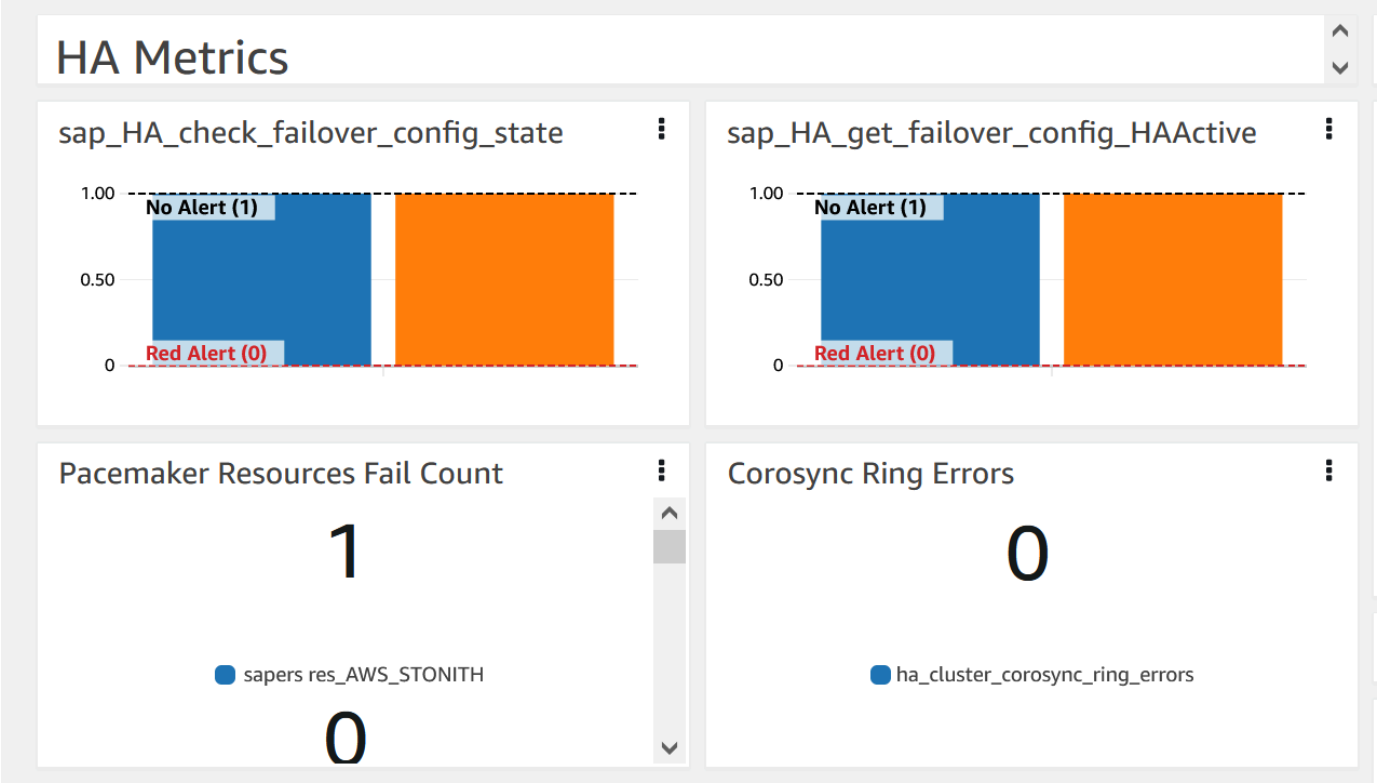


请按照以下步骤获取有关问题控制面板中显示的指标和告警的更多信息。

获取有关指标和告警的更多信息

1. 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，选择 Insights 下的 Application Insights。然后，选择 List view（列表视图）选项卡，并选择您的应用程序。
3. 选择 Components（组件）选项卡。然后，选择您想要获取更多相关信息的 SAP NetWeaver 组件。

   以下示例显示了 HA Metrics（HA 指标）部分，其中包含问题控制面板中显示的 `ha_cluster_pacemaker_fail_count` 指标。



解决方案

Application Insights 按小时监测检测到的问题。如果 SAP NetWeaver 日志文件中没有新的相关日志条目，则较旧的日志条目将被视为已解决。您必须修复与此问题相关的任何错误条件。

对于 `sap_alerts_Shortdumps` 告警，您必须使用事务代码 `RZ20 # R3Abap # Shortdumps` 导航到 CCMS 警报，以解决 SAP NetWeaver 系统中的警报。有关 CCMS 警报的更多信息，请参阅 SAP 网站。请解决 Shortdumps 树中的所有 CCMS 警报。在 SAP NetWeaver 系统中解决所有警报后，CloudWatch 将不再报告处于告警状态的指标。

解决所有 CloudWatch 日志错误和告警后，Application Insights 将停止检测错误，问题将在一小时内自动解决。我们建议您解决所有日志错误条件和告警，这样您就可以在问题控制面板上看到最新的问题。以下示例中解决了 SAP NetWeaver 高可用性入队复制问题。

Severity	Problem summary	Source	Start time	Status
High	SAP Availability: Enqueue Replication	netweavercomponent-HE2-2b8c0...	2022-12-08T20:01:43Z	Resolved

排查 Application Insights for SAP NetWeaver 的问题

本节提供了帮助您解决 Application Insights 控制面板返回的常见错误的步骤。

无法添加超过 60 个监控指标

返回错误：`Component cannot have more than 60 monitored metrics.`

根本原因：`The current metric limit is 60 monitor metrics per component.`

解决方法：删除遵守限制所不需要的指标。

SAP 指标在载入过程后未显示在控制面板上

根本原因：组件控制面板使用五分钟的指标周期来聚合数据点。

解决方法：所有指标应在五分钟后显示在控制面板上。

SAP 指标和告警未显示在控制面板上

请按照以下步骤来确定为什么 SAP 指标和告警在载入过程后未显示在控制面板上。

使用指标和告警来确定问题

1. 打开 CloudWatch 控制台。
2. 在左侧导航窗格中，选择 Insights 下的 Application Insights。然后，选择 List view（列表视图）选项卡，并选择您的应用程序。
3. 选择 Configuration history（配置历史记录）选项卡。
4. 如果您发现缺少指标数据点，请检查与 `prometheus-sap_host_exporter` 相关的错误。
5. 如果您在上一步中没有发现错误，请连接到 Linux 实例。对于高可用性部署，请连接到主集群 Amazon EC2 实例。
6. 在您的实例中，使用以下命令验证导出器是否正在运行。默认端口为 `9680`。如果您使用的是其他端口，请将 `9680` 替换为您正在使用的端口。

   ```
   curl localhost:9680/metrics
   ```

   如果没有返回数据，则导出器启动失败。
7. 要查找在接下来的两个步骤用于 `WORKLOAD_SHORT_NAME` 的正确命名约定，请运行以下命令。

> **Note**
>
> Application Insights 根据正在运行的工作负载在服务名称中添加后缀 `WORKLOAD_SHORT_NAME`。NetWeaver 分布式、标准和高可用性部署的简称分别为 `SAP_NWD`、`SAP_NWS` 和 `SAP_NWH`。

```
sudo systemctl | grep exporter
```

8. 要检查导出器服务日志中是否存在错误，请运行以下命令：

```
sudo journalctl -e --unit=prometheus-sap_host_exporter_WORKLOAD_SHORT_NAME.service
```

9. 要检查导出器管理器服务日志中是否存在错误，请运行以下命令：

```
sudo journalctl -e --unit=prometheus-
sap_host_exporter_manager_WORKLOAD_SHORT_NAME.service
```

> **Note**
>
> 此服务应始终处于启动和运行状态。

如果此命令未返回错误，则继续执行下一步。

10. 要手动启动导出器，请运行以下命令。然后，检查导出器的输出。

```
sudo /opt/aws/sap_host_exporter/sap_host_exporter
```

检查错误后，可以退出导出器进程。

根本原因：有多种可能的原因会导致此问题。一个常见的原因是导出器无法连接到其中一个应用程序服务器实例。

解决方法

请按照以下步骤将导出器连接到应用程序服务器实例。您将验证 SAP 应用程序实例是否正在运行，并使用 SAPControl 连接到该实例。

将导出器连接到应用程序服务器实例

1. 在 Amazon EC2 实例中，运行以下命令验证 SAP 应用程序是否正在运行。

```
sapcontrol -nr <App_InstNo> -function GetProcessList
```

2. 您必须建立有效的 SAPControl 连接。如果 SAPControl 连接不起作用，请在相关的 SAP 应用程序实例上找到问题的根本原因。
3. 如要在修复 SAPControl 连接问题后手动启动导出器，请运行以下命令：

```
sudo systemctl start prometheus-sap_host_exporter.service
```

4. 如果您无法解决 SAPControl 连接问题，请使用以下步骤作为临时修复方案。
   a. 打开AWS Systems Manager控制台。
   b. 从左侧导航窗格中，选择 State Manager（状态管理器）。
   c. 在 Associations（关联）下搜索 SAP NetWeaver 系统的关联。

```
Association Name: Equal: AWS-ApplicationInsights-
SSMSAPHostExporterAssociationForCUSTOMSAPNW<SID>-1
```

   d. 选择 Association id（关联 ID）。
   e. 选择 Parameters（参数）选项卡，然后从 additionalArguments（附加参数）中删除应用程序服务器编号。
   f. 选择 Apply Association Now（立即应用关联）。

> **Note**
>
> 这是临时修复。如果对组件的监控配置进行了更新，则将重新添加该实例。

查看和排查 Amazon CloudWatch Application Insights 检测到的问题

本节中的主题介绍 Application Insights 显示的检测到的问题相关详细信息和见解，并针对检测到的您账户或配置的问题提供了建议解决方法。

故障排除主题

- CloudWatch 控制台概览

- Application Insights 问题摘要页面
- CloudWatch 代理合并冲突失败
- 未创建警报
- 反馈
- 配置错误

CloudWatch 控制台概览

CloudWatch 控制台概览页面中的 CloudWatch Application Insights 窗格下列出了影响已监控应用程序的问题概览。有关更多信息，请参阅 开始使用 Amazon CloudWatch Application Insights。

CloudWatch Application Insights 概览窗格显示以下内容：

- 所检测到的问题的严重性：高/中/低
- 问题的简短摘要
- 问题来源
- 问题的开始时间
- 问题的解决状态
- 受影响的资源组

要查看特定问题的详细信息，请在 Problem Summary（问题摘要）下选择问题描述。详细控制面板显示问题信息以及相关的指标异常情况和日志错误片段。您可以选择该信息是否有用，以提供相关性反馈。

如果检测到未配置的新资源，问题摘要描述将转到 Edit configuration（编辑配置）向导以配置新资源。您可以选择详细控制面板右上角的 View/edit configuration（查看/编辑配置）来查看或编辑资源组配置。

要返回到概览，请选择 Back to overview（返回概览），它位于 CloudWatch Application Insights 详细控制面板标题旁边。

Application Insights 问题摘要页面

Application Insights 问题摘要页面

CloudWatch Application Insights 在问题摘要页面上提供有关检测到的问题的以下信息：

- 问题的简短摘要
- 问题的开始时间和日期
- 问题严重性：高/中/低
- 检测到的问题的状态：正在进行/已解决
- 信息：自动生成有关检测到的问题和可能的根本原因的信息
- 信息反馈：您为有关 CloudWatch Application Insights 生成的信息是否有用提供的反馈
- 相关的观察结果：与各种应用程序组件中的问题相关的指标异常情况和相关日志错误片段的详细视图

CloudWatch 代理合并冲突失败

CloudWatch Application Insights 在客户实例上安装和配置 CloudWatch 代理。这包括创建包含指标或日志配置的 CloudWatch 代理配置文件。如果客户的实例已经有一个 CloudWatch 代理配置文件，该文件为相同的指标或日志定义了不同的配置，则可能会发生合并冲突。要解决合并冲突，请使用下面的步骤：

1. 识别系统上的 CloudWatch 代理配置文件。有关文件位置的更多信息，请参阅 CloudWatch 代理文件和位置。
2. 从现有 CloudWatch 代理配置文件中删除要在 Application Insights 中使用的资源配置。如果您只想使用 Application Insights 配置，请删除现有的 CloudWatch 代理配置文件。

未创建警报

对于某些指标，Application Insights 会根据该指标的先前数据点预测警报阈值。如需启用此预测，必须满足以下条件。

- 最近的数据点 – 过去 24 小时内必须至少有 100 个数据点。数据点不需要是连续的，可以分散在 24 小时的时间范围内。
- 历史数据 - 从当前日期前 15 天到当前日期前 1 天的时间范围内，必须至少有 100 个数据点。数据点不需要是连续的，可以分散在 15 天的时间范围内。

> **Note**
>
> 对于某些指标，Application Insights 会延迟创建警报，直到满足上述条件。在这种情况下，您会收到一个配置历史事件，表明该指标缺乏足够的数据点来建立警报阈值。

反馈

反馈

您可以指定为检测到的问题自动生成的信息是否有用以提供反馈。将使用您对这些信息的反馈以及应用程序诊断（指标异常情况和日志异常情况）改进将来对类似问题的检测。

配置错误

CloudWatch Application Insights 使用您的配置为组件创建监控遥测数据。在 Application Insights 检测到您的账户或配置存在问题时，将在应用程序摘要的 Remarks（备注）字段中提供有关如何解决应用程序配置问题的信息。

下表显示了特定备注的建议解决方案。

备注	建议的解决方案	附加说明
已达到 CloudFormation 的配额。	Application Insights 为每个应用程序创建一个 CloudFormation 堆栈，以管理所有应用程序组件的 CloudWatch 代理安装和配置。默认情况下，每个 AWS 账户可以具有 2000 个堆栈。请参阅 AWS CloudFormation 限制。要解决该问题，请提高 CloudFormation 堆栈的限制。	不适用
在以下实例上没有 SSM 实例角色。	要使 Application Insights 能够在应用程序实例上安装和配置 CloudWatch 代理，必须将 AmazonSSMManagedInstanceCore 和 CloudWatchAgentServerPolicy 策略附加到该实例角色。	Application Insights 会调用 SSM DescribeInstanceInformation API 以获取具有 SSM 权限的实例列表。在将该角色附加到实例后，SSM 需要一段时间以将实例包含在 DescribeInstanceInformation 结果中。在 SSM 将实例包含在结果中之前，应用程序仍然存在 NO_SSM_INSTANCE_ROLE 错误。

备注	建议的解决方案	附加说明
新组件可能需要进行配置。	Application Insights 检测到在应用程序资源组中具有新组件。要解决该问题，请相应地配置新组件。	不适用

Amazon CloudWatch Application Insights 支持的日志和指标

以下列表显示 Amazon CloudWatch Application Insights 支持的日志和指标。

CloudWatch Application Insights 支持以下日志：

- Microsoft Internet 信息服务 (IIS) 日志
- EC2 上的 SQL Server 的错误日志
- 自定义 .NET 应用程序日志，例如 Log4Net
- Windows 事件日志，包括 Windows 日志（系统、应用程序和安全），以及应用程序和服务日志
- 适用于 AWS Lambda 的 Amazon CloudWatch Logs
- EC2 上 RDS MySQL、Aurora MySQL 和 MySQL 的错误日志和慢速日志
- 适用于 EC2 上的 PostgreSQL RDS 和 PostgreSQL 的 Postgresql 日志
- 适用于 AWS Step Functions 的 Amazon CloudWatch Logs
- API Gateway REST API 阶段的执行日志和访问日志（JSON、CSV 和 XML，但不包括 CLF）
- Prometheus JMX Exporter 日志 (EMF)
- Amazon RDS 上的 Oracle 和 Amazon EC2 上的 Oracle 的提示日志和侦听器日志
- 使用 `awslogs` 日志驱动程序从 Amazon ECS 容器路由到 CloudWatch 的容器日志。
- 使用 FireLens 容器日志路由器从 Amazon ECS 容器路由到 CloudWatch 的容器日志。
- 使用带有 Container Insights 的 Fluent Bit 或 Fluentd 日志处理器，从 Amazon EKS 或在 Amazon EC2 上运行的 Kubernetes 路由到 CloudWatch 的容器日志。
- SAP HANA 跟踪和错误日志
- HA Pacemaker 日志
- SAP ASE 服务器日志
- SAP ASE 备份服务器日志
- SAP ASE 复制服务器日志

- SAP ASE RMA 代理日志
- SAP ASE 故障管理器日志
- SAP NetWeaver 开发人员跟踪日志
- 使用 CloudWatch 代理的 proctstat 插件的 Windows 进程的进程指标
- 托管区域的公有 DNS 查询日志
- Amazon Route 53 Resolver DNS 查询日志

CloudWatch Application Insights 支持以下日志类：

- 标准：Amazon CloudWatch Application Insights 要求日志组配置 CloudWatch Logs 标准日志类才能启用监控。

CloudWatch Application Insights 支持以下应用程序组件的指标：

- Amazon Elastic Compute Cloud (EC2)
  - CloudWatch 内置指标
  - CloudWatch 代理指标 (Windows Server)
  - CloudWatch 代理进程指标 (Windows Server)
  - CloudWatch 代理指标 (Linux Server)
- Elastic Block Store (EBS)
- Amazon Elastic File System (Amazon EFS)
- Elastic Load Balancer (ELB)
- Application ELB
- Amazon EC2 Auto Scaling 组
- Amazon Simple Queue Server (SQS)
- Amazon Relational Database Service (RDS)
  - RDS 数据库实例
  - RDS 数据库集群
- AWS Lambda 函数
- Amazon DynamoDB 表
- Amazon S3 存储桶
- AWS Step Functions

  - Execution-level
  - 活动
  - Lambda 函数
  - 服务集成
  - Step Functions API
- API Gateway REST API 阶段
- SAP HANA
- SAP ASE
- Amazon EC2 上的 SAP ASE 高可用性
- SAP NetWeaver
- HA 集群
- Java
- Amazon Elastic Container Service (Amazon ECS)
  - CloudWatch 内置指标
  - Container Insights 指标
  - Container Insights Prometheus 指标
- AWS 上的 Kubernetes
  - Container Insights 指标
  - Container Insights Prometheus 指标
- Amazon FSx
- Amazon VPC
- Amazon VPC NAT 网关
- Amazon Route 53 运行状况检查
- Amazon Route 53 托管区
- Amazon Route 53 Resolver 端点
- AWS Network Firewall 规则组
- AWS Network Firewall 规则组关联
- 具有数据点要求的指标
  - AWS/ApplicationELB
  - AWS/AutoScaling

- AWS/EC2
- Elastic Block Store (EBS)
- AWS/ELB
- AWS/RDS
- AWS/Lambda
- AWS/SQS
- AWS/CWAgent
- AWS/DynamoDB
- AWS/S3
- AWS/States
- AWS/ApiGateway
- AWS/SNS
- 推荐的指标
- 性能计数器指标

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights 支持以下指标：

指标

- CloudWatch 内置指标
- CloudWatch 代理指标 (Windows Server)
- CloudWatch 代理进程指标 (Windows Server)
- CloudWatch 代理指标 (Linux Server)

CloudWatch 内置指标

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

磁盘读取字节数

磁盘读取操作

磁盘写入字节数

磁盘写入操作

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

网络输出

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch 代理指标 (Windows Server)

.NET CLR Exceptions # of Exceps Thrown

.NET CLR Exceptions # of Exceps Thrown/Sec

.NET CLR Exceptions # of Filters/sec

.NET CLR Exceptions # of Finallys/sec

.NET CLR Exceptions Throw to Catch Depth/sec

.NET CLR Interop # of CCWs

.NET CLR Interop # of Stubs

.NET CLR Interop # of TLB exports/sec

.NET CLR Interop # of TLB imports/sec

.NET CLR Interop # of marshaling

.NET CLR Jit % Time in Jit

.NET CLR Jit Standard Jit Failures

.NET CLR Loading % Time Loading

.NET CLR Loading Rate of Load Failures

.NET CLR LocksAndThreads Contention Rate/sec

.NET CLR LocksAndThreads Queue Length/sec

.NET CLR Memory # Total Committed Bytes

.NET CLR Memory % Time in GC

.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time

.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec

APP_POOL_WAS Total Worker Process Ping Failures

ASP.NET Application Restarts

ASP.NET Applications % Managed Processor Time (estimated)

ASP.NET Applications Errors Total/Sec

ASP.NET Applications Errors Unhandled During Execution/sec

ASP.NET Applications Requests in Application Queue

ASP.NET Applications Requests/Sec

ASP.NET Request Wait Time

ASP.NET Requests Queued

HTTP Service Request Queues CurrentQueueSize

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Memory Pages/sec

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg。Disk Queue Length

PhysicalDisk Avg。Disk sec/Read

PhysicalDisk Avg。Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Full Scans/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:General Statistics Processes blocked

SQLServer:General Statistics User Connections

SQLServer:Latches Average Latch Wait Time (ms)

SQLServer:Locks Average Wait Time (ms)

SQLServer:Locks Lock Timeouts/sec

SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec

SQLServer:Memory Manager Memory Grants Pending

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

W3SVC_W3WP File Cache Flushes

W3SVC_W3WP File Cache Misses

W3SVC_W3WP Requests/Sec

W3SVC_W3WP URI Cache Flushes

W3SVC_W3WP URI Cache Misses

Web Service Bytes Received/Sec

Web Service Bytes Sent/Sec

Web Service Connection attempts/sec

Web Service Current Connections

Web Service Get Requests/sec

Web Service Post Requests/sec

Bytes Received/sec

Normal Messages Queue Length/sec

Urgent Message Queue Length/sec

Reconnect Count

Unacknowledged Message Queue Length/sec

Messages Outstanding

Messages Sent/sec

Database Update Messages/sec

Update Messages/sec

Flushes/sec

Crypto Checkpoints Saved/sec

Crypto Checkpoints Restored/sec

Registry Checkpoints Restored/sec

Registry Checkpoints Saved/sec

Cluster API Calls/sec

Resource API Calls/sec

Cluster Handles/sec

Resource Handles/sec

CloudWatch 代理进程指标 (Windows Server)

进程指标使用 CloudWatch 代理 procstat 插件收集。只有运行 Windows 工作负载的 Amazon EC2 实例支持进程指标。

procstat cpu_time_system

procstat cpu_time_user

procstat cpu_usage

procstat memory_rss

procstat memory_vms

procstat read_bytes

procstat write_bytes

.procstat read_count

procstat write_count

CloudWatch 代理指标 (Linux Server)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used

disk_used_percent

diskio_io_time

diskio_iops_in_progress

diskio_read_bytes

diskio_read_time

diskio_reads

diskio_write_bytes

diskio_write_time

diskio_writes

mem_active

mem_available

mem_available_percent

mem_buffered

mem_cached

mem_free

mem_inactive

mem_used

mem_used_percent

net_bytes_recv

net_bytes_sent

net_drop_in

net_drop_out

net_err_in

net_err_out

net_packets_recv

net_packets_sent

netstat_tcp_close

netstat_tcp_close_wait

netstat_tcp_closing

netstat_tcp_established

netstat_tcp_fin_wait1

netstat_tcp_fin_wait2

netstat_tcp_last_ack

netstat_tcp_listen

netstat_tcp_none

netstat_tcp_syn_recv

netstat_tcp_syn_sent

netstat_tcp_time_wait

netstat_udp_socket

processes_blocked

processes_dead

processes_idle

processes_paging

processes_running

processes_sleeping

processes_stopped

processes_total

processes_total_threads

processes_wait

processes_zombies

swap_free

swap_used

swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights 支持以下指标：

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

Amazon Elastic File System (Amazon EFS)

CloudWatch Application Insights 支持以下指标：

BurstCreditBalance

PercentIOLimit

PermittedThroughput

MeteredIOBytes

TotalIOBytes

DataWriteIOBytes

DataReadIOBytes

MetadataIOBytes

ClientConnections

TimeSinceLastSync

StorageBytes

吞吐量

PercentageOfPermittedThroughputUtilization

ThroughputIOPS

PercentThroughputDataReadIOByte

PercentThroughputDataWriteIOBytes

PercentageOfIOPSDataReadIOBytes

PercentageOfIOPSDataWriteIOBytes

AverageDataReadIOBytesSize

AverageDataWriteIOBytesSize

Elastic Load Balancer (ELB)

CloudWatch Application Insights 支持以下指标：

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

RequestCount

UnHealthyHostCount

Application ELB

CloudWatch Application Insights 支持以下指标：

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

延迟

RequestCount

SurgeQueueLength

UnHealthyHostCount

Amazon EC2 Auto Scaling 组

CloudWatch Application Insights 支持以下指标：

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

磁盘读取字节数

磁盘读取操作

磁盘写入字节数

磁盘写入操作

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

网络输出

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights 支持以下指标：

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights 支持以下指标：

指标

- RDS 数据库实例
- RDS 数据库集群

RDS 数据库实例

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

读取 IOPS

ReadLatency

读取吞吐量

写入 IOPS

WriteLatency

写入吞吐量

RDS 数据库集群

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

死锁数

DeleteLatency

DeleteThroughput

EngineUptime

FreeLocalStorage

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

查询

ResultSetCacheHitRatio

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

AWS Lambda 函数

CloudWatch Application Insights 支持以下指标：

错误

DeadLetterErrors

持续时间

节流

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB 表

CloudWatch Application Insights 支持以下指标：

SystemErrors

UserErrors

已使用读取容量单位

已使用写入容量单位

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 存储桶

CloudWatch Application Insights 支持以下指标：

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS Step Functions

CloudWatch Application Insights 支持以下指标：

指标

- Execution-level
- 活动

- Lambda 函数
- 服务集成
- Step Functions API

Execution-level

ExecutionTime

ExecutionThrottled

ExecutionsFailed

ExecutionsTimedOut

ExecutionsAborted

ExecutionsSucceeded

ExecutionsStarted

活动

ActivityRunTime

ActivityScheduleTime

ActivityTime

ActivitiesFailed

ActivitiesHeartbeatTimedOut

ActivitiesTimedOut

ActivitiesScheduled

ActivitiesSucceeded

ActivitiesStarted

Lambda 函数

LambdaFunctionRunTime

LambdaFunctionScheduleTime

LambdaFunctionTime

LambdaFunctionsFailed

LambdaFunctionsTimedOut

LambdaFunctionsScheduled

LambdaFunctionsSucceeded

LambdaFunctionsStarted

服务集成

ServiceIntegrationRunTime

ServiceIntegrationScheduleTime

ServiceIntegrationTime

ServiceIntegrationsFailed

ServiceIntegrationsTimedOut

ServiceIntegrationsScheduled

ServiceIntegrationsSucceeded

ServiceIntegrationsStarted

Step Functions API

ThrottledEvents

ProvisionedBucketSize

ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API 阶段

CloudWatch Application Insights 支持以下指标：

4XXError

5XXError

IntegrationLatency

延迟

CacheHitCount

CacheMissCount

SAP HANA

> **Note**
>
> CloudWatch Application Insights 仅支持单个 SID HANA 环境。如果连接了多个 HANA SID，则只会为第一个检测到的 SID 设置监控。

CloudWatch Application Insights 支持以下指标：

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write _rate_ratio_%

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count

hanadb_failed_io_writes_count

hanadb_disk_data_unused_percent

hanadb_current_allocation_limit_used_percent

hanadb_table_allocation_limit_used_percent

hanadb_host_total_physical_memory_mb

hanadb_host_physical_memory_used_mb

hanadb_host_physical_memory_free_mb

hanadb_swap_memory_free_mb

hanadb_swap_memory_used_mb

hanadb_host_allocation_limit_mb

hanadb_host_total_memory_used_mb

hanadb_host_total_peak_memory_used_mb

hanadb_host_total_allocation_limit_mb

hanadb_host_code_size_mb

hanadb_host_shared_memory_allocation_mb

hanadb_cpu_usage_percent

hanadb_cpu_user_percent

hanadb_cpu_system_percent

hanadb_cpu_waitio_percent

hanadb_cpu_busy_percent

hanadb_cpu_idle_percent

hanadb_long_delta_merge_count

hanadb_unsuccessful_delta_merge_count

hanadb_successful_delta_merge_count

hanadb_row_store_allocated_size_mb

hanadb_row_store_free_size_mb

hanadb_row_store_used_size_mb

hanadb_temporary_tables_count

hanadb_large_non_compressed_tables_count

hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds

hanadb_average_commit_time_milliseconds

hanadb_suspended_sql_statements_count

hanadb_plan_cache_hit_ratio_percent

hanadb_plan_cache_lookup_count

hanadb_plan_cache_hit_count

hanadb_plan_cache_total_execution_microseconds

hanadb_plan_cache_cursor_duration_microseconds

hanadb_plan_cache_preparation_microseconds

hanadb_plan_cache_evicted_count

hanadb_plan_cache_evicted_microseconds

hanadb_plan_cache_evicted_preparation_count

hanadb_plan_cache_evicted_execution_count

hanadb_plan_cache_evicted_preparation_microseconds

hanadb_plan_cache_evicted_cursor_duration_microseconds

hanadb_plan_cache_evicted_total_execution_microseconds

hanadb_plan_cache_evicted_plan_size_mb

hanadb_plan_cache_count

hanadb_plan_cache_preparation_count

hanadb_plan_cache_execution_count

hanadb_network_collision_rate

hanadb_network_receive_rate

hanadb_network_transmit_rate

hanadb_network_packet_receive_rate

hanadb_network_packet_transmit_rate

hanadb_network_transmit_error_rate

hanadb_network_receive_error_rate

hanadb_time_until_license_expires_days

hanadb_is_license_valid_status

hanadb_local_running_connections_count

hanadb_local_idle_connections_count

hanadb_remote_running_connections_count

hanadb_remote_idle_connections_count

hanadb_last_full_data_backup_age_days

hanadb_last_data_backup_age_days

hanadb_last_log_backup_age_hours

hanadb_failed_data_backup_past_7_days_count

hanadb_failed_log_backup_past_7_days_count

hanadb_oldest_backup_in_catalog_age_days

hanadb_backup_catalog_size_mb

hanadb_hsr_replication_status

hanadb_hsr_log_shipping_delay_seconds

hanadb_hsr_secondary_failover_count

hanadb_hsr_secondary_reconnect_count

hanadb_hsr_async_buffer_used_mb

hanadb_hsr_secondary_active_status

hanadb_handle_count

hanadb_ping_time_milliseconds

hanadb_connection_count

hanadb_internal_connection_count

hanadb_external_connection_count

hanadb_idle_connection_count

hanadb_transaction_count

hanadb_internal_transaction_count

hanadb_external_transaction_count

hanadb_user_transaction_count

hanadb_blocked_transaction_count

hanadb_statement_count

hanadb_active_commit_id_range_count

hanadb_mvcc_version_count

hanadb_pending_session_count

hanadb_record_lock_count

hanadb_read_count

hanadb_write_count

hanadb_merge_count

hanadb_unload_count

hanadb_active_thread_count

hanadb_waiting_thread_count

hanadb_total_thread_count

hanadb_active_sql_executor_count

hanadb_waiting_sql_executor_count

hanadb_total_sql_executor_count

hanadb_data_write_size_mb

hanadb_data_write_time_milliseconds

hanadb_log_write_size_mb

hanadb_log_write_time_milliseconds

hanadb_data_read_size_mb

hanadb_data_read_time_milliseconds

hanadb_log_read_size_mb

hanadb_log_read_time_milliseconds

hanadb_data_backup_write_size_mb

hanadb_data_backup_write_time_milliseconds

hanadb_log_backup_write_size_mb

hanadb_log_backup_write_time_milliseconds

hanadb_mutex_colision_count

hanadb_read_write_lock_collision_count

hanadb_admission_control_admit_count

hanadb_admission_control_reject_count

hanadb_admission_control_queue_size_mb

hanadb_admission_control_wait_time_milliseconds

SAP ASE

CloudWatch Application Insights 支持以下指标：

asedb_database_availability

asedb_trunc_log_on_chkpt_enabled

asedb_last_db_backup_age_in_days

asedb_last_transaction_log_backup_age_in_hours

asedb_suspected_database

asedb_db_space_usage_percent

asedb_db_log_space_usage_usage_percent

asedb_locked_login

asedb_has_mixed_log_and_data

asedb_runtime_for_open_transactime

asedb_data_cache_hit_ratio

asedb_data_cache_usage_usage

asedb_sql_cache_hit_ratio

asedb_cache_usage

asedb_run_queue_length

asedb_number_of_rollbacks

asedb_of_commit_numbits

asedb_number_of_transactions

asedb_outstanding_disk_io

asedb_percent_io_busy

asedb_percent_system_busy

asedb_percent_locks_active

asedb_scheduled_jobs_failed_percent

asedb_user_connections_pection

asedb_query_logical_reads

asedb_query_physical_reads

asedb_query_cpu_time

asedb_query_memory_usage

Amazon EC2 上的 SAP ASE 高可用性

CloudWatch Application Insights 支持以下指标：

asedb_ha_replication_state

asedb_ha_replication_mode

asedb_ha_replication_latency_in_minus

SAP NetWeaver

CloudWatch Application Insights 支持以下指标：

指标	描述
sap_alerts_ResponseTime	来自 CCMS (RZ20)>R3Services>Dialog>ResponseTime 的 SAP 响应时间警报。
sap_alerts_ResponseTimeDialog	来自 CCMS (RZ20)>R3Services>Dialog>ResponseTimeDialog 的 SAP 响应时间对话警报。
sap_alerts_ResponseTimeDialogRFC	来自 CCMS (RZ20)>R3Services>Dialog>ResponseTimeDialogRFC 的 SAP 响应时间警报。
sap_alerts_DBRequestTime	来自 CCMS (RZ20)>R3Services>Dialog>DBRequestTime 的 SAP 响应时间警报。
sap_alerts_FrontendResponseTime	来自 CCMS (RZ20)>R3Services>Dialog>FrontEndResponseTime 的 SAP 响应时间警报。
sap_alerts_Database	SAP 系统记录了数据库相关错误。来自 SM21 或 CCMS (RZ20)>R3Syslog>Database 的警报。
sap_alerts_QueueTime	来自 CCMS (RZ20)>R3Services>Dialog>QueueTime 的 SAP 队列时间警报。
sap_alerts_AbortedJobs	SAP 系统中的后台作业失败。来自 (RZ20)>R3Services>Background>AbortedJobs 的警报。
sap_alerts_BasisSystem	SAP 系统记录了系统级错误。来自 SM21 或 CCMS (RZ20)>R3Syslog>BasisSystem 的警报。

指标	描述
sap_alerts_Security	SAP 系统记录了与安全相关的消息。来自 SM21 或 CCMS (RZ20)>R3Syslog>Security 的警报。
sap_alerts_System	SAP 系统记录了与安全或审核相关的消息。来自 SM21 或 CCMS (RZ20)>Security>System 的警报。
sap_alerts_LongRunners	您的 SAP 系统中有长时间运行的程序。来自 CCMS (RZ20)>R3Services>Dialog>LongRunners 的警报。
sap_alerts_SqlError	有 SAP 数据库客户端层错误日志。来自 CCMS (RZ20)>DatabaseClient>AbapSql>SqlError 的警报。
sap_alerts_State	来自 CCMS (RZ20)>OS Collector>State 的状态警报。
sap_alerts_Shortdumps	来自 ST22 和 CCMS (RZ20)>R3Abap>Shortdumps 的 Shortdumps 警报。
sap_alerts_Availability	来自 SM21、SM50、SM51、SM66 和 CCMS (RZ20)>InstanceAsTask>Availability 的 SAP 应用程序服务器实例可用性警报。
sap_dispatcher_queue_high	SAPControl Web 服务函数 `GetQueueStatistic` 提供调度程序队列的高计数。
sap_dispatcher_queue_max	SAPControl Web 服务函数 `GetQueueStatistic` 提供调度程序队列的最大计数。
sap_dispatcher_queue_now	SAPControl Web 服务函数 `GetQueueStatistic` 提供调度程序队列的当前计数。
sap_dispatcher_queue_reads	SAPControl Web 服务函数 `GetQueueStatistic` 提供调度程序队列的读取计数。

指标	描述
sap_dispatcher_queue_writes	SAPControl Web 服务函数 GetQueueStatistic 提供调度程序队列的写入计数。
sap_enqueue_server_arguments_high	SAPControl Web 服务函数 EnqGetStatistic 提供高入队参数。
sap_enqueue_server_arguments_max	SAPControl Web 服务函数 EnqGetStatistic 提供最大入队参数。
sap_enqueue_server_arguments_now	SAPControl Web 服务函数 EnqGetStatistic 提供当前入队参数。
sap_enqueue_server_arguments_state	SAPControl Web 服务函数 EnqGetStatistic 提供入队参数状态。
sap_enqueue_server_backup_requests	SAPControl Web 服务函数 EnqGetStatistic 提供入队备份请求。
sap_enqueue_server_cleanup_requests	SAPControl Web 服务函数 EnqGetStatistic 提供入队清理请求。
sap_enqueue_server_dequeue_all_requests	SAPControl Web 服务函数 EnqGetStatistic 提供全部出队请求。
sap_enqueue_server_dequeue_errors	SAPControl Web 服务函数 EnqGetStatistic 提供出队错误。
sap_enqueue_server_dequeue_requests	SAPControl Web 服务函数 EnqGetStatistic 提供出队请求。
sap_enqueue_server_enqueue_errors	SAPControl Web 服务函数 EnqGetStatistic 提供入队错误。
sap_enqueue_server_enqueue_rejects	SAPControl Web 服务函数 EnqGetStatistic 提供入队拒绝。
sap_enqueue_server_enqueue_requests	SAPControl Web 服务函数 EnqGetStatistic 提供入队请求。

指标	描述
sap_enqueue_server_lock_time	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定时间。
sap_enqueue_server_lock_wait_time	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定等待时间。
sap_enqueue_server_locks_high	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定的高计数。
sap_enqueue_server_locks_max	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定的最大计数。
sap_enqueue_server_locks_now	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定的当前计数。
sap_enqueue_server_locks_state	SAPControl Web 服务函数 EnqGetStatistic 提供入队锁定状态。
sap_enqueue_server_owner_high	SAPControl Web 服务函数 EnqGetStatistic 提供入队所有者的高计数。
sap_enqueue_server_owner_max	SAPControl Web 服务函数 EnqGetStatistic 提供入队所有者的最大计数。
sap_enqueue_server_owner_now	SAPControl Web 服务函数 EnqGetStatistic 提供入队所有者的当前计数。
sap_enqueue_server_owner_state	SAPControl Web 服务函数 EnqGetStatistic 提供入队所有者状态。
sap_enqueue_server_replication_state	SAPControl Web 服务函数 EnqGetStatistic 提供入队复制状态。
sap_enqueue_server_reporting_requests	SAPControl Web 服务函数 EnqGetStatistic 提供报告请求状态。
sap_enqueue_server_server_time	SAPControl Web 服务函数 EnqGetStatistic 提供入队服务器时间。

指标	描述
sap_HA_check_failover_config_state	SAPControl Web 服务函数 `HACheckFailoverConfig` 提供 SAP 高可用性状态。
sap_HA_get_failover_config_HAActive	SAPControl Web 服务函数 `HAGetFailoverConfig` 提供 SAP 高可用性集群配置和状态。
sap_start_service_processes	SAPControl Web 服务函数 `GetProcessList` 提供 disp+work、IGS、gwrd、icman、消息服务器和入队服务器进程状态。

HA 集群

CloudWatch Application Insights 支持以下指标：

ha_cluster_pacemaker_stonith_enabled

ha_cluster_corosync_quorate

hanadb_webdispatcher_service_started_status

ha_cluster_pacemaker_nodes

ha_cluster_corosync_ring_errors

ha_cluster_pacemaker_fail_count

Java

CloudWatch Application Insights 支持以下指标：

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_threading_daemonthreadcount

java_lang_classloading_loadedclasscount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights 支持以下指标：

指标

- CloudWatch 内置指标
- Container Insights 指标
- Container Insights Prometheus 指标

CloudWatch 内置指标

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights 指标

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity

instance_memory_utilization

instance_memory_working_set

instance_network_total_bytes

instance_number_of_running_tasks

Container Insights Prometheus 指标

Java JMX 指标

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

AWS 上的 Kubernetes

CloudWatch Application Insights 支持以下指标：

指标

- Container Insights 指标
- Container Insights Prometheus 指标

Container Insights 指标

cluster_failed_node_count

cluster_node_count

namespace_number_of_running_pods

node_cpu_limit

node_cpu_reserved_capacity

node_cpu_usage_total

node_cpu_utilization

node_filesystem_utilization

node_memory_limit

node_memory_reserved_capacity

node_memory_utilization

node_memory_working_set

node_network_total_bytes

node_number_of_running_containers

node_number_of_running_pods

pod_cpu_reserved_capacity

pod_cpu_utilization

pod_cpu_utilization_over_pod_limit

pod_memory_reserved_capacity

pod_memory_utilization

pod_memory_utilization_over_pod_limit

pod_network_rx_bytes

pod_network_tx_bytes

service_number_of_running_pods

Container Insights Prometheus 指标

Java JMX 指标

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights 支持以下指标：

DataReadBytes

DataWriteBytes

DataReadOperations

DataWriteOperations

MetadataOperations

FreeStorageCapacity

FreeDataStorageCapacity

LogicalDiskUsage

PhysicalDiskUsage

Amazon VPC

CloudWatch Application Insights 支持以下指标：

NetworkAddressUsage

NetworkAddressUsagePeered

VPCFirewallQueryVolume

Amazon VPC NAT 网关

CloudWatch Application Insights 支持以下指标：

ErrorPortAllocation

IdleTimeoutCount

Amazon Route 53 运行状况检查

CloudWatch Application Insights 支持以下指标：

ChildHealthCheckHealthyCount

ConnectionTime

HealthCheckPercentageHealthy

HealthCheckStatus

SSLHandshakeTime

TimeToFirstByte

Amazon Route 53 托管区

CloudWatch Application Insights 支持以下指标：

DNSQueries

DNSSECInternalFailure

DNSSECKeySigningKeysNeedingAction

DNSSECKeySigningKeyMaxNeedingActionAge

DNSSECKeySigningKeyAge

Amazon Route 53 Resolver 端点

CloudWatch Application Insights 支持以下指标：

EndpointHealthyENICount

EndpointUnHealthyENICount

InboundQueryVolume

OutboundQueryVolume

OutboundQueryAggregateVolume

AWS Network Firewall 规则组

CloudWatch Application Insights 支持以下指标：

FirewallRuleGroupQueryVolume

AWS Network Firewall 规则组关联

CloudWatch Application Insights 支持以下指标：

FirewallRuleGroupVpcQueryVolume

具有数据点要求的指标

对于没有触发警报的明显默认阈值的指标，Application Insights 等到指标具有足够的数据点以预测触发警报的合理阈值。CloudWatch Application Insights 在告警创建之前检查的指标数据点要求如下：

- 指标从过去 15 天到过去 2 天期间至少具有 100 个数据点。
- 指标在最后一天至少具有 100 个数据点。

以下指标遵循这些数据点要求。请注意，CloudWatch 代理指标最多需要一小时才能创建警报。

指标

- AWS/ApplicationELB
- AWS/AutoScaling
- AWS/EC2
- Elastic Block Store (EBS)
- AWS/ELB
- AWS/RDS
- AWS/Lambda
- AWS/SQS
- AWS/CWAgent
- AWS/DynamoDB
- AWS/S3
- AWS/States
- AWS/ApiGateway
- AWS/SNS

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS/AutoScaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

磁盘读取字节数

磁盘读取操作

磁盘写入字节数

磁盘写入操作

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

网络输出

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

延迟

RequestCount

SurgeQueueLength

UnHealthyHostCount

AWS/RDS

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

CPUCreditBalance

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

死锁数

DeleteLatency

DeleteThroughput

DiskQueueDepth

EngineUptime

FreeLocalStorage

FreeStorageSpace

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

查询

读取 IOPS

读取吞吐量

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

写入 IOPS

写入吞吐量

AWS/Lambda

错误

DeadLetterErrors

持续时间

节流

IteratorAge

ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg。Disk sec/Read

PhysicalDisk Avg。Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer：接收的数据库副本文件字节数/秒

SQLServer：接收的数据库副本日志字节数/秒

SQLServer：保留数据库副本日志以供撤消

SQLServer：数据库副本日志发送队列

SQLServer：数据库副本镜像写事务数/秒

SQLServer：数据库副本恢复队列

SQLServer：剩余数据库副本重做字节数

SQLServer：数据库副本已重做的字节数/秒

SQLServer：需要撤消的数据库副本总日志

SQLServer：数据库副本事务延迟

SQLServer:General Statistics Processes blocked

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

AWS/DynamoDB

已使用读取容量单位

已使用写入容量单位

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

AWS/S3

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS/States

ActivitiesScheduled

ActivitiesStarted

ActivitiesSucceeded

ActivityScheduleTime

ActivityRuntime

ActivityTime

LambdaFunctionsScheduled

LambdaFunctionsStarted

LambdaFunctionsSucceeded

LambdaFunctionScheduleTime

LambdaFunctionRuntime

LambdaFunctionTime

ServiceIntegrationsScheduled

ServiceIntegrationsStarted

ServiceIntegrationsSucceeded

ServiceIntegrationScheduleTime

ServiceIntegrationRuntime

ServiceIntegrationTime

ProvisionedRefillRate

ProvisionedBucketSize

ConsumedCapacity

ThrottledEvents

AWS/ApiGateway

4XXError

IntegrationLatency

延迟

DataProcessed

CacheHitCount

CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered

NumberOfMessagesPublished

NumberOfNotificationsFailed

NumberOfNotificationsFilteredOut

NumberOfNotificationsFilteredOut-InvalidAttributes

NumberOfNotificationsFilteredOut-NoMessageAttributes

NumberOfNotificationsRedrivenToDlq

NumberOfNotificationsFailedToRedriveToDlq

SMSSuccessRate

推荐的指标

下表列出了每种组件类型的建议指标。

组件类型	工作负载类型	建议的指标
EC2 实例 (Windows Server)	默认/自定义	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Active Directory	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes 数据库 ==> 实例数据库缓存 % 命中 DirectoryServices DRA 挂起的复制操作 DirectoryServices DRA 挂起的复制同步

组件类型	工作负载类型	建议的指标
		DNS 递归查询失败/秒 LogicalDisk 平均值 Disk Queue Length
	Java 应用程序	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

组件类型	工作负载类型	建议的指标
	Microsoft IIS/.NET Web 前端	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts

组件类型	工作负载类型	建议的指标
	Microsoft SQL Server 数据库层	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Paging File % Usage System Processor Queue Length Network Interface Bytes Total/Sec PhysicalDisk % Disk Time SQLServer:Buffer Manager Buffer Cache Hit ratio SQLServer:Buffer Manager Page Life Expectancy SQLServer:General Statistics Processes Blocked SQLServer:General Statistics User Connections SQLServer:Locks Number of Deadlocks/Sec SQLServer:SQL Statistics Batch Requests/Sec

组件类型	工作负载类型	建议的指标
	MySQL	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	.NET 工作线程池/中间层	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC

组件类型	工作负载类型	建议的指标
	.NET Core 层	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes
	Oracle	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Postgres	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes

组件类型	工作负载类型	建议的指标
	SharePoint	CPUUtilization
		StatusCheckFailed
		Processor % Processor Time
		Memory % Committed Bytes In Use
		Memory Available Mbytes
		ASP.NET 应用程序缓存 API 修剪
		ASP.NET 请求被拒
		ASP.NET 工作进程重启
		Memory Pages/sec
		SharePoint 发布缓存发布缓存刷新/秒
		SharePoint Foundation 执行时间/页面请求
		SharePoint 基于磁盘的缓存缓存压缩总数
		SharePoint 基于磁盘的缓存 Blob 缓存命中率
		SharePoint 基于磁盘的缓存 Blob 缓存填充率
		SharePoint 基于磁盘的缓存 Blob 缓存刷新/秒
		ASP.NET Requests Queued

组件类型	工作负载类型	建议的指标
		ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk 平均值 Disk sec/Write LogicalDisk 平均值 Disk sec/Read Processor % Interrupt Time
EC2 实例 (Linux Server)	默认/自定义	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

组件类型	工作负载类型	建议的指标
	Java 应用程序	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
	.NET Core 层或 SQL Server 数据库层	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

组件类型	工作负载类型	建议的指标
	Oracle	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
	Postgres	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

组件类型	工作负载类型	建议的指标
EC2 实例组	SAP HANA 多节点或单节点	• hanadb_server_startup_time_variations_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_rate_ratio_% • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

组件类型	工作负载类型	建议的指标
EBS 卷	任何	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	任何	HTTPCode_Backend_4XX HTTPCode_Backend_5XX 延迟 SurgeQueueLength UnHealthyHostCount
Application ELB	任何	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount

组件类型	工作负载类型	建议的指标
RDS 数据库实例	任何	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS 数据库集群	任何	CPUUtilization CommitLatency DatabaseConnections 死锁数 FreeableMemory NetworkThroughput VolumeBytesUsed
Lambda 函数	任何	持续时间 错误 IteratorAge ProvisionedConcurrencySpilloverInvocations 节流

组件类型	工作负载类型	建议的指标
SQS Queue	任何	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB 表	任何	SystemErrors UserErrors 已使用读取容量单位 已使用写入容量单位 ReadThrottleEvents WriteThrottleEvents ConditionalCheckFailedRequests TransactionConflict

组件类型	工作负载类型	建议的指标
Amazon S3 存储桶	任何	如果启用了带有复制时间控制 (RTC) 的复制配置： ReplicationLatency BytesPendingReplication OperationsPendingReplication 如果启用了请求指标： 5xxErrors 4xxErrors BytesDownloaded BytesUploaded

组件类型	工作负载类型	建议的指标
AWS Step Functions	任何	常规 • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity 如果状态机类型为 **EXPRESS** 或日志组级别为 **OFF** • ExecutionsFailed • ExecutionsTimedOut 如果状态机有 Lambda 函数 • LambdaFunctionsFailed • LambdaFunctionsTimedOut 如果状态机有活动 • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbeatTimedOut 如果状态机有服务集成 • ServiceIntegrationsFailed • ServiceIntegrationsTimedOut

组件类型	工作负载类型	建议的指标
API Gateway REST API 阶段	任何	• 4xxErrors • 5xxErrors • 延迟

组件类型	工作负载类型	建议的指标
ECS 集群	任何	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation（仅限 EC2 启动类型） CPUUtilization（仅限 EC2 启动类型） MemoryReservation（仅限 EC2 启动类型） MemoryUtilization（仅限 EC2 启动类型） GPUReservation（仅限 EC2 启动类型） instance_cpu_utilization（仅限 EC2 启动类型） instance_filesystem_utilization（仅限 EC2 启动类型） instance_memory_utilization（仅限 EC2 启动类型）

组件类型	工作负载类型	建议的指标
		instance_network_total_bytes（仅限 EC2 启动类型）

组件类型	工作负载类型	建议的指标
	Java 应用程序	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation（仅限 EC2 启动类型） CPUUtilization（仅限 EC2 启动类型） MemoryReservation（仅限 EC2 启动类型） MemoryUtilization（仅限 EC2 启动类型） GPUReservation（仅限 EC2 启动类型） instance_cpu_utilization（仅限 EC2 启动类型） instance_filesystem_utilization（仅限 EC2 启动类型） instance_memory_utilization（仅限 EC2 启动类型）

组件类型	工作负载类型	建议的指标
		instance_network_total_bytes（仅限 EC2 启动类型） java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
ECS 服务	任何	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes

组件类型	工作负载类型	建议的指标
	Java 应用程序	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

组件类型	工作负载类型	建议的指标
EKS 集群	任何	cluster_failed_node_count
		node_cpu_reserved_capacity
		node_cpu_utilization
		node_filesystem_utilization
		node_memory_reserv ed_capacity
		node_memory_utilization
		node_network_total_bytes
		pod_cpu_reserved_capacity
		pod_cpu_utilization
		pod_cpu_utilization_over_po d_limit
		pod_memory_reserve d_capacity
		pod_memory_utilization
		pod_memory_utilization_over _pod_limit
		pod_network_rx_bytes
		pod_network_tx_bytes

组件类型	工作负载类型	建议的指标
	Java 应用程序	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

组件类型	工作负载类型	建议的指标
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

组件类型	工作负载类型	建议的指标
EC2 上的 Kubernetes 集群	任何	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

组件类型	工作负载类型	建议的指标
	Java 应用程序	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

组件类型	工作负载类型	建议的指标
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

下表列出了每种组件类型的建议流程和流程指标。CloudWatch Application Insights 不建议对不在实例上运行的流程进行流程监控。

组件类型	工作负载类型	建议的流程	建议的指标

EC2 实例 (Windows Server)	Microsoft IIS/.NET Web 前端	`w3wp`	`procstat cpu_usage ,`
			
			`procstat memory_rss ,`
			
			`procstat memory_vms ,`
			
			`procstat read_bytes ,`
			
			`procstat write_bytes`

	Microsoft SQL Server 数据库层	`SQLAgent`	`procstat cpu_usage ,`
			
			`procstat memory_rss ,`

组件类型	工作负载类型	建议的流程	建议的指标
			procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlservr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlwriter	procstat cpu_usage , procstat memory_rss
		Reporting ServicesS ervice	procstat cpu_usage , procstat memory_rss

组件类型	工作负载类型	建议的流程	建议的指标
		MsDtsServr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		Msmdsrv	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

组件类型	工作负载类型	建议的流程	建议的指标
	.NET 工作线程池/中间层	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	.NET Core 层	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

性能计数器指标

仅当 Windows 实例上安装了相应的性能计数器集时，才建议对实例使用性能计数器指标。

性能计数器指标名称	性能计数器集名称
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions

性能计数器指标名称	性能计数器集名称
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions
.NET CLR Exceptions Throw to Catch Depth/Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0

性能计数器指标名称	性能计数器集名称
.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET 请求被拒	ASP.NET
ASP.NET 工作进程重启	ASP.NET
ASP.NET 应用程序缓存 API 修剪	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET
数据库 ==> 实例数据库缓存 % 命中	数据库 ==> 实例

性能计数器指标名称	性能计数器集名称
数据库 ==> 实例 I/O 数据库读取平均延迟	数据库 ==> 实例
数据库 ==> 实例 I/O 数据库读取/秒	数据库 ==> 实例
数据库 ==> 实例 I/O 日志写入平均延迟	数据库 ==> 实例
DirectoryServices DRA 挂起的复制操作	DirectoryServices
DirectoryServices DRA 挂起的复制同步	DirectoryServices
DirectoryServices LDAP 绑定时间	DirectoryServices
DNS 递归查询/秒	DNS
DNS 递归查询失败/秒	DNS
DNS TCP 接收查询/秒	DNS
DNS 总接收查询/秒	DNS
DNS 总发送响应/秒	DNS
DNS UDP 接收查询/秒	DNS
HTTP Service Request Queues CurrentQueueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk 平均值 Disk sec/Write	LogicalDisk
LogicalDisk 平均值 Disk sec/Read	LogicalDisk
LogicalDisk 平均值 Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	内存
Memory Available Mbytes	内存
Memory Pages/Sec	内存

性能计数器指标名称	性能计数器集名称
内存长期平均备用缓存寿命（秒）	内存
Network Interface Bytes Total/Sec	网络接口
网络接口接收的字节数/秒	网络接口
网络接口发送的字节数/秒	网络接口
网络接口当前带宽	网络接口
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg。Disk Queue Length	PhysicalDisk
PhysicalDisk Avg。Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg。Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	处理器
Processor % Interrupt Time	处理器
Processor % Processor Time	处理器
Processor % User Time	处理器
SharePoint 基于磁盘的缓存 Blob 缓存填充率	SharePoint 基于磁盘的缓存
SharePoint 基于磁盘的缓存 Blob 缓存刷新/秒	SharePoint 基于磁盘的缓存

性能计数器指标名称	性能计数器集名称
SharePoint 基于磁盘的缓存 Blob 缓存命中率	SharePoint 基于磁盘的缓存
SharePoint 基于磁盘的缓存缓存压缩总数	SharePoint 基于磁盘的缓存
SharePoint Foundation 执行时间/页面请求	SharePoint Foundation
SharePoint 发布缓存发布缓存刷新/秒	SharePoint 发布缓存
安全系统范围的统计信息 Kerberos 身份验证	安全系统范围的统计
安全系统范围的统计信息 NTLM 身份验证	安全系统范围的统计
SQLServer:Access Methods Forwarded Records/Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager
SQLServer:Buffer Manager Page life Expectancy	SQLServer:Buffer Manager
SQLServer：接收的数据库副本文件字节数/秒	SQLServer：数据库副本
SQLServer：接收的数据库副本日志字节数/秒	SQLServer：数据库副本
SQLServer：保留数据库副本日志以供撤消	SQLServer：数据库副本
SQLServer：数据库副本日志发送队列	SQLServer：数据库副本
SQLServer：数据库副本镜像写事务数/秒	SQLServer：数据库副本
SQLServer：数据库副本恢复队列	SQLServer：数据库副本
SQLServer：剩余数据库副本重做字节数	SQLServer：数据库副本
SQLServer：数据库副本已重做的字节数/秒	SQLServer：数据库副本

性能计数器指标名称	性能计数器集名称
SQLServer：需要撤消的数据库副本总日志	SQLServer：数据库副本
SQLServer：数据库副本事务延迟	SQLServer：数据库副本
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compilations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	系统
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP

性能计数器指标名称	性能计数器集名称
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	Web 服务
Web Service Bytes Sent/Sec	Web 服务
Web Service Connection Attempts/Sec	Web 服务
Web Service Current Connections	Web 服务
Web Service Get Requests/Sec	Web 服务
Web Service Post Requests/Sec	Web 服务

使用 CloudWatch 控制台中的资源运行状况视图

您可以使用资源运行状况视图在单个视图中自动发现、管理和可视化主机在其所有应用程序中的运行状况和性能。您可以按性能维度（如 CPU 或内存）可视化显示其主机的运行状况，并使用筛选条件在单个视图中对数百台主机进行切片和切块。您可以按标签或使用案例进行筛选，例如同一 Auto Scaling 组中的主机或使用同一负载均衡器的主机，

先决条件

要确保充分享受资源运行状况视图的益处，请检查您是否满足以下先决条件。

- 要查看主机的内存利用率并将其用作筛选条件，您必须在主机上安装 CloudWatch 代理，并将其设置为在默认 `CWAgent` 命名空间中向 CloudWatch 发送内存指标。在 Linux 和 macOS 实例上，CloudWatch 代理必须发送 `mem_used_percent` 指标。在 Windows 实例上，此代理必须发送 `Memory % Committed Bytes In Use` 指标。如果您使用向导创建 CloudWatch 代理配置文件并选择任意预定义的指标集，则这些指标会包含在其中。CloudWatch 代理收集的指标按自定义指标计费。有关更多信息，请参阅安装 CloudWatch 代理。

当您使用 CloudWatch 代理收集这些内存指标以用于资源运行状况视图时，您必须将以下部分添加到 CloudWatch 代理配置文件中。此部分包含默认维度设置，并且是预设情况下的创建内容，因此请勿更改此部分的任何内容，保持其与以下示例所示内容一致。

```
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
```

- 要查看资源运行状况视图中的所有可用信息，您必须登录具有以下权限的账户。如果您登录的账户的权限较少，您仍然可以使用资源运行状况视图，但某些性能数据将不可见。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "sns:Get*",
                "sns:List*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeRegions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

在账户中查看资源运行状况

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择基础设施监控、资源运行状况。

   此时将显示资源运行状况页面，页面上显示账户中每台主机对应的一个方块。根据 Color by（颜色依据）设置，每个方块的颜色都基于该主机的当前状态。主机方块带有告警符号，表示该主机当前有一个或多个处于 ALARM 状态的告警。

   您可以在单个视图中查看多达 500 台主机的运行状况。如果您账户中主机数量超过此数量，请使用此程序步骤 6 中的筛选条件设置。
3. 若要更改显示每个主机运行状况所基于的条件，请选择 Color by（颜色依据）。您可以选择 CPU Utilization（CPU 利用率）、Memory Utilization（内存利用率），或 Status check（状态检查）。内存利用率指标仅适用于运行 CloudWatch 代理且将其配置为收集内存指标并将指标发送到默认 `CWAgent` 命名空间的主机。有关更多信息，请参阅使用 CloudWatch 代理收集指标、日志和跟踪信息。
4. 若要更改用于网格中运行状况指标的阈值和颜色，请选择网格上方的齿轮图标。
5. 若要切换是否在主机网格中显示告警，请选择或清除 Show alarms across all metrics（显示所有指标的告警）。
6. 若要将地图中的主机拆分为组，请选择 Group by（分组依据）。
7. 若要将视图所涵盖的主机范围缩小，请选择 Filter by（筛选条件）中的筛选条件。您可以按标签和资源分组（如 Auto Scaling 组、实例类型、安全组等）进行筛选。
8. 若要对主机进行排序，请选择 Sort by（排序方式）。您可以按状态检查结果、实例状态、CPU 或内存利用率以及处于 ALARM 状态的告警数量进行排序。
9. 要查看有关主机的详细信息，请选择表示该主机的方块。随后会显示一个弹出窗格。然后要深入了解有关该主机的信息，请选择 View dashboard（查看控制面板）或 View on list（在列表中查看）。

跨账户和跨区域监控

为实现跨账户统一监控，CloudWatch 提供了以下功能：

- CloudWatch 跨账户可观测性：使用 Observability Access Manager（OAM）服务促进单个区域内的可观测性。您可以关联账户，轻松查看账户之间的指标、日志、跟踪和其他遥测数据。这有助于您统一中央监控账户中的可观测性，这些账户可以查看从源账户共享的遥测数据，并像对监控账户的原生遥测数据一样对这种共享遥测数据进行操作。
- 跨账户跨区域 CloudWatch 控制台：提供控制台体验，允许您通过在账户之间切换来查看跨区域其他账户的控制面板、指标和警报控制台。设置必要的权限后，您可以使用集成到警报、控制面板和指标控制台中的账户选择器来查看其他账户中的指标、控制面板和警报，而无需登录和注销账户。启用此功能后，您还可以设置包含跨账户跨区域指标的控制面板，以便在账户内集中查看。

这两项功能相互补充，可以单独使用，也可以一起使用。请参阅下表，了解两项功能的对比。建议您使用 CloudWatch 跨账户可观测性，以得到区域内指标、日志和跟踪记录最丰富的跨账户可观测性和发现体验。

	CloudWatch 跨账户可观测性	跨账户跨区域的 CloudWatch 控制台
这是什么？	跨多个账户统一访问底层遥测和其他可观测性资源。配置完成后，便可以在账户之间无缝查看可观测性资源，而无需担任角色。中央监控账户可以直接访问源账户的遥测数据和资源，从而简化了监控和可观测性流程。	指定的监控账户担任在 CloudWatch 控制台源账户中定义的 CrossAccountSharingRole。担任此角色后，监控账户可以直接从其控制台调用代表源账户查看控制面板等操作。
如何工作？	使用可观测性访问监控服务的监控账户会创建一个接收器并向其附加接收器策略。接收器策略定义接收器想要查看哪些资源，以及哪些源账户应该共享这些资源。然后，源账户可以创建指向监控账户接收器的链接，确定其实际想要共享的内	源账户通过设置 CrossAccountSharingRole 来启动配置，从而允许监控账户在源账户中运行操作。然后，监控账户通过指定源账户 ID 在控制台中启用跨账户跨区域选择器。这使监控账户能够切换到源账户。切换时，CloudWatch 控制台会检查是否存在服务相关

	CloudWatch 跨账户可观测性	跨账户跨区域的 CloudWatch 控制台
	容。创建链接后，指定的资源将在监控账户中显示。	角色，该角色允许 CloudWatch 担任在源账户中创建的 CrossAccountSharingRole。
支持哪些遥测？	• 指标 • 跟踪 • 日志 .	• 指标 • 跟踪
支持哪些功能？	• 控制面板 • 告警 • 指标洞察 • 异常检测 • CloudWatch Logs Insights • Application Insights • 其他功能。有关更多详细信息，请参阅CloudWatch 跨账户可观测性。 .	• 在指标、警报和跟踪控制台中，在账户和区域之间切换控制台。 • 包含来自其他账户和区域的指标和警报的自定义控制面板 有关更多详细信息，请参阅跨账户跨区域的 CloudWatch 控制台。
我可以将其与多少个账户一起使用？	一个监控账户可以同时查看多达 10 万个源账户的资源。一个源账户最多可以与五个不同的监控账户共享其资源。	使用控制台中的跨账户跨区域选择器后，一个监控账户一次只能切换到一个其他账户，但可以关联的账户数量没有限制。定义跨账户控制面板和警报时，可以引用多个源账户。
会移动遥测数据吗？	不会。除了复制的跟踪以外，资源在账户之间共享。	不会。IAM 策略配置为允许切换嵌入式账户以实现跨账户跨区域资源可见性。

	CloudWatch 跨账户可观测性	跨账户跨区域的 CloudWatch 控制台
它的成本是多少？	共享日志和指标无需额外付费，并且第一个跟踪副本免费。有关定价的更多信息，请参见 Amazon CloudWatch 定价。	跨账户或跨区域操作无需额外付费。
是否支持跨区域的可观测性？	否	是
是否支持编程访问？	是。支持 AWS CLI、AWS Cloud Development Kit (AWS CDK) 和 API。	否。
是否支持编程设置？	是	是
是否支持 AWS Organizations？	是	是

主题

- CloudWatch 跨账户可观测性
- 跨账户跨区域的 CloudWatch 控制台

CloudWatch 跨账户可观测性

通过 Amazon CloudWatch 跨账户可观测性，您可以监控跨越一个区域内多个账户的应用程序并对其进行故障排除。无缝地搜索、可视化显示和分析任何关联账户中的指标、日志、跟踪记录、Application Insights 应用程序和网络监测仪，而不受账户限制。

将一个或多个 AWS 账户设置为监控账户，并将它们与多个源账户相关联。监控账户是一个中央 AWS 账户，可以查看源账户生成的可观测性数据并与之交互。源账户是一个单独的 AWS 账户，可为其中的资源生成可观测性数据。源账户与监控账户共享其可观测性数据。共享的可观测性数据可以包括以下类型的遥测数据：

- Amazon CloudWatch 中的指标。您可以选择与监控账户共享所有命名空间的指标，也可以筛选为命名空间的子集。

- Amazon CloudWatch Logs 中的日志组。您可以选择与监控账户共享所有日志组，也可以筛选为日志组的子集。
- AWS X-Ray 中的跟踪记录
- Amazon CloudWatch Application Insights 中的应用程序
- CloudWatch 网络监测仪中的监测仪

要在监控账户和源账户之间创建关联，您可以使用 CloudWatch 控制台。或者，使用 AWS CLI 和 API 中的 Observability Access Manager 命令。有关更多信息，请参阅 Observability Access Manager API 参考。

接收器是表示监控账户中连接点的资源。源账户可以与接收器相关联，以共享可观测性数据。每个账户在每个区域可以有一个接收器。每个接收器都由其所在的监控账户管理。可观测性关联是一种资源，表示在源账户和监控账户之间建立的关联。关联由源账户管理。

有关设置 CloudWatch 跨账户可观测性的视频演示，请观看以下视频。

下一个主题介绍如何在监控账户和源账户中设置 CloudWatch 跨账户可观测性。有关跨账户跨区域 CloudWatch 控制面板的信息，请参阅 跨账户跨区域的 CloudWatch 控制台。

使用 Organizations 作为源账户

将源账户关联到监控账户有两个选项。您可以使用其中一个选项，或同时使用两个选项。

- 使用 AWS Organizations 将组织或组织单位中的账户关联到监控账户。
- 将单独的 AWS 账户连接到监控账户。

我们建议您使用 Organizations，这样稍后在组织中创建的新 AWS 账户就会自动作为源账户载入到跨账户可观测性中。

关于关联监控账户和源账户的详细信息

- 每个监控账户可以关联多达 10 万个源账户。
- 每个源账户最多可以与五个监控账户共享数据。
- 您可以将单个账户同时设置为监控账户和源账户。如果您这样做，则此账户仅将其自身的可观测性数据发送到其关联的监控账户。它不会转发来自其源账户的数据。
- 监控账户指定可以与其共享的遥测类型。源账户指定要共享的遥测类型。

  - 如果在监控账户中选择的遥测类型多于在源账户中选择的遥测类型，则会关联这些账户。只有在两个账户中都选择的数据类型才会进行共享。
  - 如果在源账户中选择的遥测类型多于在监控账户中选择的遥测类型，则关联创建失败，不会共享任何内容。
  - 在创建链接后该指标发出新数据点之前，指标名称不会在监控账户控制台中显示。
- 要删除账户之间的关联，请从源账户执行此操作。
- 要删除监控账户中的接收器，必须先删除与该监控账户中接收器的所有关联。

定价

CloudWatch 中的跨账户可观测性没有额外的日志和指标成本，而且第一个跟踪副本是免费的。有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

目录

- 将监控账户与源账户相关联
  - 必要的权限
  - 设置概述
  - 步骤 1：设置监控账户
  - 步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL
  - 步骤 3：关联源账户
    - 使用 AWS CloudFormation 模板将组织或组织单位中的所有账户设置为源账户
    - 使用 AWS CloudFormation 模板设置单个源账户
    - 使用 URL 设置单个源账户
- 管理监控账户和源账户
  - 将更多源账户关联到现有的监控账户
  - 删除监控账户与源账户之间的关联
  - 查看监控账户的相关信息

将监控账户与源账户相关联

本节中的主题介绍了如何在监控账户和源账户之间设置关联。

我们建议您创建一个新的 AWS 账户作为贵组织的监控账户。

目录

- 必要的权限
- 设置概述
- 步骤 1：设置监控账户
- 步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL
- 步骤 3：关联源账户
  - 使用 AWS CloudFormation 模板将组织或组织单位中的所有账户设置为源账户
  - 使用 AWS CloudFormation 模板设置单个源账户
  - 使用 URL 设置单个源账户

必要的权限

如要在监控账户和源账户之间创建关联，您必须以特定权限登录。

- 设置监控账户 – 您必须拥有监控账户的完全管理员访问权限，或者使用以下权限登录该账户：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSinkModification",
            "Effect": "Allow",
            "Action": [
                "oam:CreateSink",
                "oam:DeleteSink",
                "oam:PutSinkPolicy",
                "oam:TagResource"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnly",
            "Effect": "Allow",
            "Action": ["oam:Get*", "oam:List*"],
            "Resource": "*"
        }
    ]
}
```

- 源账户，范围限定为特定的监控账户 – 如要仅为一个指定的监控账户创建、更新和管理关联，您必须至少使用以下权限登录该账户。在本示例中，监控账户为 999999999999。

  如果此关联未打算共享所有五种资源类型（指标、日志、跟踪、Application Insights 应用程序和网络检测仪监控），则可以根据需要省略 `cloudwatch:Link`、`logs:Link`、`xray:Link`、`applicationinsights:Link` 或 `internetmonitor:Link`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink",
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:link/*"
        },
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:sink/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": [
                        "999999999999"
                    ]
                }
            }
        },
        {
            "Action": "oam:ListLinks",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
```

```
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
         },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
    ]
}
```

- 源账户，具有关联到任何监控账户的权限 – 要创建与任何现有监控账户接收器的关联并共享指标、日志组、跟踪、Application Insights 应用程序和网络监测仪，您必须以完全管理员权限登录源账户或使用以下权限登录

  如果此关联未打算共享所有五种资源类型（指标、日志、跟踪、Application Insights 应用程序和网络检测仪监控），则可以根据需要省略 `cloudwatch:Link`、`logs:Link`、`xray:Link`、`applicationinsights:Link` 或 `internetmonitor:Link`。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
```

```
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:List*",
                "oam:Get*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
```

```
            "Action": "internetmonitor:Link",
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

设置概述

以下高级步骤向您展示了如何设置 CloudWatch 跨账户可观测性。

> **Note**
>
> 我们建议创建一个新的 AWS 账户，用作贵组织的监控账户。

1. 设置一个专用的监控账户。
2. （可选）下载 AWS CloudFormation 模板或复制 URL 以关联源账户。
3. 将源账户关联到该监控账户。

完成这些步骤后，您可以使用该监控账户查看源账户的可观测性数据。

步骤 1：设置监控账户

按照本节中的步骤将 AWS 账户设置为 CloudWatch 跨账户可观测性的监控账户。

先决条件

- 如果您将 AWS Organizations 组织中的账户设置为源账户 – 获取组织路径或组织 ID。
- 如果您不使用 Organizations 作为源账户 – 获取源账户的账户 ID。

要将一个账户设置为监控账户，您必须具有特定的权限。有关更多信息，请参阅 必要的权限。

设置监控账户

1. 登录要用作监控账户的账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在左侧导航窗格中，选择 设置。

4. 通过 Monitoring account configuration（监控账户配置），选择 Configure（配置）。
5. 对于选择数据，选择该监控账户是否能够查看所关联源账户的日志、指标、跟踪、Application Insights – 应用程序和网络监测仪 - 监控数据。
6. 对于 List source accounts（列出源账户），输入该监控账户将查看的源账户。要标识源账户，请输入单个账户 ID、组织路径或组织 ID。如果您输入组织路径或组织 ID，则该监控账户可以查看该组织中所有关联账户的可观测性数据。

   用逗号分隔此列表中的条目。

   > **Important**
   >
   > 输入组织路径时，请遵循确切的格式。ou-id 必须以 /（斜杠字符）结尾。例如：o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbb/

7. 对于 Define a label to identify your source account（定义一个标签来标识源账户），请指定在使用监控账户查看源账户时是使用账户名还是电子邮件地址来标识源账户。
8. 选择 配置。

> **Important**
>
> 在您配置源账户之前，监控账户和源账户之间的关联是不完整的。有关更多信息，请参阅以下部分。

步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL

要将源账户关联到监控账户，我们建议使用 AWS CloudFormation 模板或 URL。

- 如果您要关联整个组织 – CloudWatch 提供了一个 AWS CloudFormation 模板。
- 如果您要关联单个账户 – 使用 AWS CloudFormation 模板或 CloudWatch 提供的 URL。

要使用 AWS CloudFormation 模板，您必须在以下步骤中下载该模板。将监控账户与至少一个源账户关联后，将不能再下载 AWS CloudFormation 模板。

下载 AWS CloudFormation 模板或复制 URL，以便将源账户关联到监控账户

1. 登录要用作监控账户的账户。

2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在左侧导航窗格中，选择 设置。
4. 通过 Monitoring account configuration（监控账户配置），选择 Resources to link accounts（用于关联账户的资源）。
5. 请执行以下操作之一：
   - 选择 AWS organization 以获取用于将组织中的账户关联到该监控账户的模板。
   - 选择 Any account（任何账户）以获取用于将单个账户设置为源账户的模板或 URL。
6. 请执行以下操作之一：
   - 如果您选择了 AWS organization，请选择 Download CloudFormation template（下载 CloudFormation 模板）。
   - 如果您选择了 Any account（任何账户），请选择 Download CloudFormation template（下载 CloudFormation 模板）或 Copy URL（复制 URL）。
7. （可选）重复步骤 5-6以下载 AWS CloudFormation 模板并复制 URL。

步骤 3：关联源账户

按照以下步骤将源账户关联到监控账户。

要将监控账户与源账户相关联，您必须具有特定的权限。有关更多信息，请参阅 必要的权限。

使用 AWS CloudFormation 模板将组织或组织单位中的所有账户设置为源账户

以下步骤假设您已经通过执行 步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL 中的步骤下载了必要的 AWS CloudFormation 模板。

使用 AWS CloudFormation 模板将组织或组织单位中的账户关联到监控账户

1. 登录到组织的管理账户。
2. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
3. 在左侧导航栏中，选择 StackSets（堆栈集）。
4. 检查您是否已登录到所需的区域，然后选择 Create StackSet（创建堆栈集）。
5. 选择下一步。
6. 选择 Template is ready（模板已准备就绪），然后选择 Upload a template file（上传模板文件）。

7. 选择 Choose file（选择文件），然后选择从监控账户下载的模板，再选择 Open（打开）。
8. 选择下一步。
9. 对于 Specify StackSet details（指定堆栈集详细信息），输入堆栈集名称并选择 Next（下一步）。
10. 对于 Add stacks to stack set（将堆栈添加到堆栈集），选择 Deploy new stacks（部署新堆栈）。
11. 对于 Deployment targets（部署目标），选择是部署到整个组织还是部署到指定的组织单位。
12. 对于 Specify regions（指定区域），选择要将 CloudWatch 跨账户可观测性部署到哪些区域。
13. 选择下一步。
14. 在 Review（审核）页面上，确认所选的选项并选择 Submit（提交）。
15. 在 Stack instances（堆栈实例）选项卡中，刷新屏幕，直到您看到堆栈实例的状态为 CREATE_COMPLETE。

使用 AWS CloudFormation 模板设置单个源账户

以下步骤假设您已经通过执行 步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL 中的步骤下载了必要的 AWS CloudFormation 模板。

使用 AWS CloudFormation 模板为 CloudWatch 跨账户可观测性设置单个源账户

1. 登录到源账户。
2. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
3. 在左侧导航栏中，选择 Stacks（堆栈）。
4. 检查您是否已登录到所需的区域，然后依次选择 Create stack（创建堆栈）、With new resources (standard)（使用新资源（标准））。
5. 选择下一步。
6. 选择上传模板文件。
7. 选择 Choose file（选择文件），然后选择从监控账户下载的模板，再选择 Open（打开）。
8. 选择下一步。
9. 对于 Specify stack details（指定堆栈详细信息），输入堆栈名称并选择 Next（下一步）。
10. 在配置堆栈选项页面上，请选择下一步。
11. 在 Review（审核）页面上，选择 Submit（提交）。
12. 在堆栈的状态页面上，刷新屏幕，直到您看到堆栈的状态为 CREATE_COMPLETE。

13. 要使用同一模板将更多源账户关联到该监控账户，请退出此源账户并登录到下一个源账户。然后重复步骤 2-12。

使用 URL 设置单个源账户

以下步骤假设您已经通过执行 步骤 2：（可选）下载 AWS CloudFormation 模板或复制 URL 中的步骤复制了必要的 URL。

使用 URL 将单个源账户关联到监控账户

1. 登录要用作源账户的账户。
2. 输入从监控账户复制的 URL。

   您会看到 CloudWatch 设置页面，其中填写了一些信息。
3. 对于选择数据，选择该源账户是否将日志、指标、跟踪、Application Insights – 应用程序和网络监测仪 - 监控数据共享到该监控账户。

   对于日志和指标，您可以选择是与监控账户共享所有资源还是共享部分资源。

   a. （可选）要与监控账户共享此账户日志组的子集，请选择日志，然后选择筛选日志。然后，使用筛选日志框构造查询以查找要共享的日志组。该查询将使用条件 `LogGroupName` 和以下一个或多个操作数。

      - `=` 和 `!=`
      - `AND`
      - `OR`
      - `^` 表示 LIKE，`!^` 表示 NOT LIKE。这些只能用作前缀搜索。在要搜索和包含的字符串末尾加上 `%`。
      - `IN` 和 `NOT IN`，使用圆括号 (( ))

      完整的查询不得超过 2000 个字符，并且限制为五个条件操作数。条件操作数为 `AND` 和 `OR`。其他操作数的数量没有限制。

      > **Tip**
      >
      > 选择查看示例查询，以查看常见查询格式的正确语法。

b. （可选）要与监控账户共享此账户指标命名空间的子集，请选择指标，然后选择筛选指标。然后，使用筛选指标框构造查询以查找要共享的指标命名空间。使用条件 Namespace 和以下一个或多个操作数。

- = 和 !=
- AND
- OR
- LIKE 和 NOT LIKE 这些只能用作前缀搜索。在要搜索和包含的字符串末尾加上 %。
- IN 和 NOT IN，使用圆括号 (( ))

完整的查询不得超过 2000 个字符，并且限制为五个条件操作数。条件操作数为 AND 和 OR。其他操作数的数量没有限制。

> **Tip**
>
> 选择查看示例查询，以查看常见查询格式的正确语法。

4. 请勿在 Enter monitoring account configuration ARN（输入监控账户配置 ARN）中更改 ARN。
5. Define a label to identify your source account（定义一个标签来标识源账户）部分预先填充了监控账户中选择的标签。或者，也可以选择 Edit（编辑）来更改标签。
6. 选择关联。
7. 在此框中输入 **Confirm**，然后选择 Confirm（确认）。
8. 要使用同一 URL 将更多源账户关联到该监控账户，请退出此源账户并登录到下一个源账户。然后重复步骤 2-7。

管理监控账户和源账户

设置监控账户和源账户后，您可以按照以下步骤对其进行管理。

目录

- 将更多源账户关联到现有的监控账户
- 删除监控账户与源账户之间的关联
- 查看监控账户的相关信息

将更多源账户关联到现有的监控账户

按照本节中的步骤将更多源账户关联到现有的监控账户。

每个源账户最多可以关联五个监控账户。每个监控账户可以关联多达 10 万个源账户。

要管理源账户，您必须具有特定的权限。有关更多信息，请参阅 必要的权限。

将更多源账户添加到监控账户

1. 登录到监控账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在左侧导航窗格中，选择 设置。
4. 通过 Monitoring account configuration（监控账户配置），选择 Manage source accounts（管理源账户）。
5. 选择 Configuration policy（配置策略）选项卡。
6. 在 Configuration policy（配置策略）框中，在 Principal（主体）一行添加新的源账户 ID。

   例如，假设 Principal（主体）一行当前的内容如下：

   ```
   "Principal": {"AWS": ["111111111111", "222222222222"]}
   ```

   要将 999999999999 添加为第三个源账户，请将该行编辑为以下内容：

   ```
   "Principal": {"AWS": ["111111111111", "222222222222", "999999999999"]}
   ```
7. 选择更新。
8. 选择 Configuration details（配置详细信息）选项卡。
9. 选择监控账户的接收器 ARN 旁边的复制图标。
10. 登录要用作新源账户的账户。
11. 粘贴您在步骤 9 中复制的监控账户的接收器 ARN。

    您会看到 CloudWatch 设置页面，其中填写了一些信息。
12. 对于选择数据，选择该源账户是否要将日志、指标、跟踪和 Application Insights – 应用程序数据发送到所关联的监控账户。
13. 请勿在 Enter monitoring account configuration ARN（输入监控账户配置 ARN）中更改 ARN。

14. Define a label to identify your source account（定义一个标签来标识源账户）部分预先填充了监控账户中选择的标签。或者，也可以选择 Edit（编辑）来更改标签。
15. 选择关联。
16. 在此框中输入 **`Confirm`**，然后选择 Confirm（确认）。

删除监控账户与源账户之间的关联

按照本节中的步骤停止从一个源账户向监控账户发送数据。

您必须拥有管理源账户所需的权限才能完成此任务。有关更多信息，请参阅 必要的权限。

删除源账户与监控账户之间的关联

1. 登录到源账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在左侧导航窗格中，选择 设置。
4. 通过 Source account information（源账户信息），选择 View monitoring accounts（查看监控账户）。
5. 选中要停止与之共享数据的监控账户旁边的复选框。
6. 依次选择 Stop sharing data（停止共享数据）、Confirm（确认）。
7. 登录到监控账户。
8. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
9. 选择设置。
10. 通过 Monitoring account information（监控账户信息），选择 View configuration（查看配置）。
11. 在 Policy（策略）框中，从 Principal（主体）一行删除源账户 ID，然后选择 Update（更新）。

查看监控账户的相关信息

按照本节中的步骤查看监控账户的跨账户设置。

要管理监控账户，您必须具有特定的权限。有关更多信息，请参阅 必要的权限。

管理监控账户

1. 登录到监控账户。

2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在左侧导航窗格中，选择 设置。
4. 通过 Monitoring account configuration（监控账户配置），选择 Manage source accounts（管理源账户）。
5. 要查看使该账户成为监控账户的 Observability Access Manager 策略，请选择 Configuration policy（配置策略）选项卡。
6. 要查看与该监控账户关联的源账户，请选择 Linked source accounts（关联的源账户）选项卡。
7. 要查看监控账户接收器 ARN 以及该监控账户可以在关联的源账户中查看的数据类型，请选择 Linked source accounts（关联的源账户）选项卡。

跨账户跨区域的 CloudWatch 控制台

> **Note**
>
> 建议您使用 CloudWatch 跨账户可观测性，以获取区域内指标、日志和跟踪记录最丰富的跨账户可观测性和发现体验。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

跨账户、跨区域 CloudWatch 控制台允许您使用控制台中的选择器在不同的账户和区域之间轻松切换，以查看其他账户和区域中的控制面板、警报和指标。此功能还可以创建跨账户、跨区域的控制面板，将来自多个 AWS 账户和多个区域的 CloudWatch 指标汇总到一个控制面板中，从而无需切换账户或区域即可访问这些指标。

许多组织都将其 AWS 资源部署到多个账户中来提供账单和安全边界。在这种情况下，建议您将一个或多个账户指定为监控账户，并在这些账户中构建跨账户跨区域控制面板。跨账户跨区域控制台功能与 AWS Organizations 集成，可帮助您高效地构建跨账户跨区域控制面板。

跨账户、跨区域 CloudWatch 控制台体验不提供跨账户跨区域的日志可见性。此外，它不支持从监控账户内创建其他账户或区域中指标的警报。

主题

- 在 CloudWatch 中启用跨账户跨区域功能
- （可选）与 AWS Organizations 集成
- 对 CloudWatch 跨账户设置问题进行故障排除
- 使用跨账户后禁用和清理

在 CloudWatch 中启用跨账户跨区域功能

要在 CloudWatch 控制台中设置跨账户跨区域功能，请使用 CloudWatch 控制台设置您的共享账户和监控账户。

设置共享账户

您必须在每个账户中启用共享，以使数据对监控账户可用。

这将向所有在您共享的账户中查看跨账户控制面板的用户授予您在步骤 5 中选择的只读权限（如果该用户在您共享的账户中具有相应权限）。

使您的账户能够与其他账户共享 CloudWatch 数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。
3. 对于 Share your CloudWatch data（共享 CloudWatch 数据），选择 Configure（配置）。
4. 对于 Sharing (共享)，选择 Specific accounts (特定账户)，然后输入要与之共享数据的账户的 ID。

   您在此处指定的任何账户都可以查看您的账户的 CloudWatch 数据。仅指定您知道并信任的账户的 ID。
5. 对于 Permissions (权限)，使用下列选项之一来指定共享数据的方式：

   - Provide read-only access to your CloudWatch metrics, dashboards, and alarms（提供对 CloudWatch 指标、控制面板和告警的只读访问权限）。此选项使监控账户能够创建跨账户控制面板，这些控制面板具有包含您账户中的 CloudWatch 数据的小组件。
   - 包括 CloudWatch 自动控制面板。如果选择此选项，则监控账户中的用户还可以查看此账户的自动控制面板中的信息。有关更多信息，请参阅 Amazon CloudWatch 入门。
   - 包含对 X-Ray 跟踪地图的 X-Ray 只读访问 如果选择此选项，监控账户中的用户还可以查看 X-Ray 跟踪地图，以及此账户中的 X-Ray 跟踪信息。有关更多信息，请参阅 Using the X-Ray Trace Map。
   - Full read-only access to everything in your account (对账户中所有内容的完全只读访问权)。此选项使您用于共享的账户能够创建跨账户控制面板，这些控制面板具有包含您账户中的 CloudWatch 数据的小组件。它还使这些账户能够更深入地查看您的账户，并在其他 AWS 服务的控制台中查看您账户的数据。
6. 选择 Launch CloudFormation template (启动 CloudFormation 模板)。

   在确认屏幕中，键入 **`Confirm`**，并选择 Launch template (启动模板)。

7. 选中 I acknowledge...(我确认...) 复选框，然后选择 Create stack (创建堆栈)。

与整个企业共享

通过完成上述过程，创建一个 IAM 角色来使您的账户能够与某个账户共享数据。您可以创建或编辑一个 IAM 角色，该角色将与企业中所有的账户共享您的数据。仅在您知道并信任组织中所有的账户的情况下执行此操作。

如果用户在与您共享的帐户中具有相应权限，这将向在您的共享账户中查看跨账户控制面板的所有用户授予在上一过程第 5 步所示策略中列出的只读权限。

与企业中所有的账户共享您的 CloudWatch 账户数据

1. 如果尚未执行此操作，请完成上述过程来与某个 AWS 账户共享您的数据。
2. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
3. 在导航窗格中，选择角色。
4. 在角色列表中，选择 CloudWatch-CrossAccountSharingRole。
5. 选择信任关系，然后选择编辑信任关系。

   您将看到与以下内容类似的策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. 将策略更改为以下内容，并将 *org-id* 替换为您组织的 ID。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "org-id"
        }
      }

    }
  ]
}
```

7. 选择更新信任策略。

设置监控帐户

如果要查看跨账户 CloudWatch 数据，请启用每个监控账户。

完成以下过程后，CloudWatch 将创建一个服务相关角色，CloudWatch 在监控账户中使用该角色来访问从其他账户共享的数据。此服务相关角色称为 AWSServiceRoleForCloudWatchCrossAccount。有关更多信息，请参阅 为 CloudWatch 使用服务相关角色。

使您的账户能够查看跨账户 CloudWatch 数据

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置），然后在 Cross-account cross-region（跨账户跨区域）部分中，选择 Configure（配置）。
3. 在 View cross-account cross-region（查看跨账户跨区域）部分，选择 Enable（启用），然后选择 Show selector in the console（在控制台中显示选择器）复选框以在您绘制指标图表或创建告警时，让账户选择器显示在 CloudWatch 控制台中。
4. 在 View cross-account cross-region (跨账户跨区域查看) 中，选择下列选项之一：

   - Account Id Input (账户 ID 输入)。在查看跨账户的数据时，每当您要切换账户时，此选项都会提示您手动输入账户 ID。

- AWS Organization account selector（Amazon Organizations 账户选择器）。此选项可显示您在完成与 Organizations 的跨账户集成时指定的账户。在您下次使用控制台时，CloudWatch 会为您显示这些账户的下拉列表，供您在查看跨账户数据时进行选择。

  为此，您必须已先使用企业管理账户允许 CloudWatch 查看组织中的账户列表。有关更多信息，请参阅 （可选）与 AWS Organizations 集成。
- Custom account selector (自定义账户选择器)。此选项提示您输入账户 ID 的列表。在您下次使用控制台时，CloudWatch 会为您显示这些账户的下拉列表，供您在查看跨账户数据时进行选择。

  此外，您可为其中每个账户输入一个标签，以便在选择要查看的账户时帮助您识别它们。

  用户在此处进行的账户选择器设置仅为该用户保留，而不为监视账户中的所有其他用户保留。

5. 请选择 启用。

完成此设置后，您可以创建跨账户控制面板。有关更多信息，请参阅 跨账户跨区域的控制面板。

跨区域功能

跨区域功能自动内置到此功能中。您无需执行任何额外步骤即可在同一图表或同一控制面板上的单个账户中显示来自不同区域的指标。由于告警不支持跨区域功能，您无法在某个区域中创建用以监视其他区域内指标的告警。

（可选）与 AWS Organizations 集成

如果要将跨账户功能与 AWS Organizations 集成，您必须使组织中的所有账户对监控账户可用。

启用跨账户 CloudWatch 功能以访问企业中的所有账户

1. 登录到企业的管理账户。
2. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
3. 在导航窗格中，选择 Settings (设置)，然后选择 Configure (配置)。
4. 对于 Grant permission to view the list of accounts in the organization (授予查看组织中账户列表的权限)，选择要提示输入账户 ID 列表的 Specific accounts (特定账户)。您组织中的账户列表仅与您在此处指定的账户共享。
5. 选择 Share organization account list (共享组织账户列表)。
6. 选择 Launch CloudFormation template (启动 CloudFormation 模板)。

在确认屏幕中，键入 **Confirm**，并选择 Launch template (启动模板)。

对 CloudWatch 跨账户设置问题进行故障排除

这部分包含有关 CloudWatch 中跨账户控制台部署的故障排除提示。

我在显示跨账户的数据时收到访问被拒绝错误

请检查以下事项：

- 您的监控账户应具有一个名为 AWSServiceRoleForCloudWatchCrossAccount 的角色。如果没有此角色，则需要创建它。有关更多信息，请参阅 Set Up a Monitoring Account。
- 每个共享账户应具有一个名为 CloudWatch-CrossAccountSharingRole 的角色。如果没有此角色，则需要创建它。有关更多信息，请参阅 Set Up A Sharing Account。
- 此共享角色必须信任监控账户。

确认是否已为 CloudWatch 跨账户控制台正确设置您的角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。
3. 在角色列表中，确保需要的角色已存在。在共享账户中，查找 CloudWatch-CrossAccountSharingRole。在监控账户中，查找 AWSServiceRoleForCloudWatchCrossAccount。
4. 如果您正在使用共享账户，并且 CloudWatch-CrossAccountSharingRole 已存在，请选择 CloudWatch-CrossAccountSharingRole。
5. 选择信任关系，然后选择编辑信任关系。
6. 确认策略已列出监控账户的账户 ID 或监控账户所属组织的组织 ID。

控制台中未显示账户下拉菜单

首先，检查您是否已创建正确的 IAM 角色，如前面的故障排除部分中所述。如果已正确设置这些角色，请确保您已启用此账户来查看跨账户的数据，如Enable Your Account to View Cross-Account Data中所述。

使用跨账户后禁用和清理

要禁用 CloudWatch 的跨账户功能，请执行下列步骤。

步骤 1：删除跨账户堆栈或角色

最佳方法是删除用于启用跨账户功能的 AWS CloudFormation 堆栈。

- 在每个共享账户中，删除 CloudWatch-CrossAccountSharingRole 堆栈。
- 如果您使用 AWS Organizations 为企业中的所有账户启用跨账户功能，请删除企业管理账户中的 CloudWatch-CrossAccountListAccountsRole 堆栈。

如果您没有使用 AWS CloudFormation 堆栈以启用跨账户功能，请执行以下操作：

- 在每个共享账户中，删除 CloudWatch-CrossAccountSharingRole IAM 角色。
- 如果您使用 AWS Organizations 为企业中的所有账户启用跨账户功能，请删除企业管理账户中的 CloudWatch-CrossAccountSharing-ListAccountsRole IAM 角色。

步骤 2：删除服务相关角色

在监控账户中，删除 AWSServiceRoleForCloudWatchCrossAccount 服务相关的 IAM 角色。

查询源自其他数据来源的指标

您可以使用 CloudWatch 对源自其他数据来源的指标进行查询、可视化并为其创建警报。为此，您需要将 CloudWatch 连接到其他数据来源。这样，您便可在 CloudWatch 控制台中进行集中统一监控。无论数据存储在何处，您都可以统一查看基础设施和应用程序指标，从而帮助您更快地发现和解决问题。

使用 CloudWatch 向导连接到数据来源后，CloudWatch 会创建一个用于部署和配置 AWS Lambda 函数的 AWS CloudFormation 堆栈。每次查询数据来源时，此 Lambda 函数都会按需运行。CloudWatch 查询构建器会实时显示可查询元素的列表，例如指标、表、字段或标签。在您做出选择后，查询构建器会以选定来源的原生语言预先填充一个查询。

您可以使用 CloudWatch 提供的引导式向导连接到以下数据来源。对于这些数据来源，您需要提供一些基本信息来识别数据来源和凭证。您还可以通过创建自己的 Lambda 函数来手动创建连接其他数据来源的连接器。

- Amazon OpenSearch Service：从您的 OpenSearch Service 日志和跟踪中派生指标。
- Amazon Managed Service for Prometheus：使用 PromQL 查询这些指标。
- Amazon RDS for MySQL：使用 SQL 将存储在 Amazon RDS 表中的数据转换为指标。
- Amazon RDS for PostgreSQL：使用 SQL 将存储在 Amazon RDS 表中的数据转换为指标。
- Amazon S3 CSV 文件：显示存储在 Amazon S3 存储桶中的 CSV 文件内的指标数据。
- Microsoft Azure Monitor：查询 Microsoft Azure Monitor 账户中的指标。
- Prometheus：使用 PromQL 查询这些指标。

创建连接到数据来源的连接器后，请参阅 创建源自另一个数据来源的指标图表 查看有关绘制来自某个数据来源的指标的图表的信息。有关为数据来源中的指标设置警报的信息，请参阅 基于连接的数据来源创建警报。

主题

- 管理对数据来源的访问
- 通过向导连接到预构建数据来源
- 创建自定义数据来源连接器。
- 使用您的自定义数据来源
- 删除数据来源连接器

管理对数据来源的访问

CloudWatch 通过 AWS CloudFormation 在您的账户中创建所需资源。在向 IAM 用户授予 `CreateStack` 权限时，我们建议您使用 `cloudformation:TemplateUrl` 条件来控制对 AWS CloudFormation 模板的访问权限。

> **Warning**
>
> 您对其授予数据来源调用权限的任何用户都可以查询该数据来源中的指标，即使此用户没有该数据来源的直接 IAM 访问权限。例如，如果您向用户授予 Amazon Managed Service for Prometheus 数据来源 Lambda 函数的 `lambda:InvokeFunction` 权限，则即使您没有向用户授予该工作区的直接 IAM 访问权限，该用户也将能够查询相应 Amazon Managed Service for Prometheus 工作区的指标。

您可以在 CloudWatch 设置控制台的创建堆栈页面上找到数据来源的模板 URL。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect" : "Allow",
      "Action" : [ "cloudformation:CreateStack" ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudformation:TemplateUrl" : [ data-source-template-url ]
        }
      }
    }
  ]
}
```

有关控制 AWS CloudFormation 访问的更多信息，请参阅使用 AWS Identity and Access Management 控制访问。

通过向导连接到预构建数据来源

本主题提供相关说明，帮助您了解如何通过向导将 CloudWatch 连接到以下数据来源。

- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon RDS for MySQL
- Amazon RDS for PostgreSQL
- Amazon S3 CSV 文件
- Microsoft Azure Monitor
- Prometheus

本节后面的若干小节中包含有关管理这些数据来源和使用其进行查询的说明。

创建数据来源连接器。

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。
3. 选择指标数据来源选项卡。
4. 选择创建数据来源。
5. 选择所需的来源，然后选择下一步。
6. 输入数据来源的名称。
7. 根据所选择的数据来源，输入其他必填信息。这些信息可能包括用于访问数据来源的凭证和数据来源标识信息，例如 Prometheus 工作区名称、数据库名称或 Amazon S3 存储桶名称。关于 AWS 服务，向导会发现资源并将其填充到选择下拉列表中。

   有关您正在使用的数据来源的更多说明，请参阅此过程之后的小节。
8. 要将 CloudWatch 连接到 VPC 中的数据来源，请选择使用 VPC，然后选择要使用的 VPC。然后选择子网和安全组。
9. 选择我确认 AWS CloudFormation 会创建 IAM 资源。此资源是 Lambda 函数执行角色。
10. 选择创建数据来源。

    AWS CloudFormation 堆栈完成创建新来源之后，您刚刚添加的新来源才会出现。要查看进度，可以选择查看我的 CloudFormation 堆栈的状态。您也可以使用刷新图标来更新此列表。

    新数据来源出现在此列表中时即可使用。您可以选择从 CloudWatch 指标中查询，以开始使用新数据来源进行查询。有关更多信息，请参阅 创建源自另一个数据来源的指标图表。

Amazon Managed Service for Prometheus

更新数据来源配置

- 您可以通过以下步骤手动更新数据来源：
  - 要更新 Amazon Managed Service for Prometheus 工作区 ID，请更新数据来源连接器 Lambda 函数的 AMAZON_PROMETHEUS_WORKSPACE_ID 环境变量。
  - 要更新 VPC 配置，请参阅配置 VPC 访问（控制台）了解更多信息。

查询数据来源

- 查询 Amazon Managed Service for Prometheus 时，在多来源查询选项卡中选择数据来源并选择一个 Amazon Managed Service for Prometheus 连接器后，您可以通过查询助手来发现指标和标签并提供简单的 PromQL 查询。您也可以使用 PromQL 查询编辑器来构建 PromQL 查询。
- CloudWatch 数据来源连接器不支持多行查询。执行查询，或者使用查询创建警报或控制面板小组件时，每个换行符都会替换为空格。在某些情况下，这可能会导致查询无效。例如，如果您的查询包含单行注释，则该查询无效。如果您尝试利用命令行或基础设施即代码中的多行查询来创建控制面板或警报，API 将拒绝该操作，并显示解析错误。

Amazon OpenSearch Service

创建数据来源

如果为 FGAC 启用 OpenSearch 域，您必须将连接器 Lambda 函数的执行角色映射到 OpenSearch Service 中的用户。有关更多信息，请参阅 OpenSearch Service 文档管理权限中的将角色映射到用户部分。

如果您的 OpenSearch 域只能在虚拟私有云（VPC）中访问，则需要在 Lambda 函数中手动包含一个名为 AMAZON_OPENSEARCH_ENDPOINT 的新环境变量。此变量的值应为 OpenSearch 端点的根域。您可以通过从 OpenSearch 服务控制台列出的域端点中移除 https:// 和 <region>.es.amazonaws.com 来获取此根域。例如，如果您的域端点是 https://sample-domain.us-east-1.es.amazonaws.com，则根域为 sample-domain。

更新数据来源

- 您可以通过以下步骤手动更新数据来源：
  - 要更新 OpenSearch Service 域，请更新数据来源连接器 Lambda 函数的 AMAZON_OPENSEARCH_DOMAIN_NAME 环境变量。

- 要更新 VPC 配置，请参阅配置 VPC 访问（控制台）了解更多信息。

查询数据来源

- 查询 OpenSearch Service 时，在多来源查询选项卡中选择数据来源后，请执行以下操作：
  - 选择要查询的索引。
  - 选择指标名称（文档中的任何数值字段）和统计数据。
  - 选择时间轴（文档中的任何日期字段）。
  - 选择要应用的筛选条件（文档中的任意字符串字段）。
  - 选择图表查询。

Amazon RDS for PostgreSQL 和 Amazon RDS for MySQL

创建数据来源

- 如果数据来源只能在 VPC 中访问，则连接器必须包含 VPC 配置，如 通过向导连接到预构建数据来源 中所述。如果数据来源要连接到 VPC 以获取凭证，则必须在 VPC 中配置端点。有关更多信息，请参阅使用 AWS Secrets Manager VPC 端点。

  此外，您必须为 Amazon RDS 服务创建 VPC 端点。有关更多信息，请参阅 Amazon RDS API 和接口 VPC 端点（AWS PrivateLink）。

更新数据来源

- 您可以通过以下步骤手动更新数据来源：
  - 要更新数据库实例，请更新数据来源连接器 Lambda 函数的 `RDS_INSTANCE` 环境变量。
  - 要更新用于连接 Amazon RDS 的用户名和密码，请使用 AWS Secrets Manager。您可以在数据来源 Lambda 函数的 `RDS_SECRET` 环境变量中找到用于数据来源的密钥的 ARN。有关更新 AWS Secrets Manager 中的密钥的更多信息，请参阅修改 AWS Secrets Manager 密钥。
  - 要更新 VPC 配置，请参阅配置 VPC 访问（控制台）了解更多信息。

查询数据来源

- 查询 Amazon RDS 时，在多来源查询选项卡中选择数据来源并选择 Amazon RDS 连接器后，即可使用数据库发现器查看可用的数据库、表和列。您还可以使用 SQL 编辑器创建 SQL 查询。

您可以在查询中使用以下变量：

- `$start.iso`：ISO 日期格式的开始时间
- `$end.iso`：ISO 日期格式的结束时间
- `$period`：选定时段（以秒为单位）

例如，您可以查询 `SELECT value, timestamp FROM table WHERE timestamp BETWEEN $start.iso and $end.iso`

- CloudWatch 数据来源连接器不支持多行查询。执行查询，或者使用查询创建警报或控制面板小组件时，每个换行符都会替换为空格。在某些情况下，这可能会导致查询无效。例如，如果您的查询包含单行注释，则该查询无效。如果您尝试利用命令行或基础设施即代码中的多行查询来创建控制面板或警报，API 将拒绝该操作，并显示解析错误。

> **Note**
>
> 如果在结果中找不到日期字段，则将每个数值字段的值相加为单个值，并绘制在所提供的时间范围内。如果时间戳与 CloudWatch 中的选定时段不一致，则系统会使用 SUM 自动聚合数据，并且数据将与 CloudWatch 中的时段保持一致。

Amazon S3 CSV 文件

查询数据来源

- 查询 Amazon S3 CSV 文件时，在多来源查询选项卡中选择数据来源并选择 Amazon S3 连接器后，即可选择 Amazon S3 存储桶和键。

CSV 文件必须通过以下方式格式化：

- 时间戳必须位于第一列。
- 该表必须有标题行。标题用于命名您的指标。时间戳列的标题将被忽略，仅使用指标列的标题。
- 时间戳必须采用 ISO 日期格式。
- 指标必须是数值字段。

```
Timestamp, Metric-1, Metric-2, ...
```

以下是 示例：

时间戳	CPU（%）	内存(%)	存储（%）
2023-11-23T17:09:41+00:00	1	2	3
2023-11-23T17:04:41+00:00	4	5	6
2023-11-23T16:59:41+00:00	7	8	9
2023-11-23T16:54:41+00:00	10	11	12

> **Note**
>
> 如果未提供时间戳，系统会将每个指标的值相加为单个值，并在所提供的时间范围内绘制。如果时间戳与 CloudWatch 中的选定时段不一致，则系统会使用 SUM 自动聚合数据，并且数据将与 CloudWatch 中的时段保持一致。

Microsoft Azure Monitor

创建数据来源

- 您必须提供租户 ID、客户端 ID 和客户端密钥才能连接到 Microsoft Azure Monitor。凭证将存储于 AWS Secrets Manager 内。有关更多信息，请参阅 Microsoft 文档中的 Create a Microsoft Entra application and service principal that can access resources。

更新数据来源

- 您可以通过以下步骤手动更新数据来源：
  - 要更新用于连接到 Azure Monitor 的租户 ID、客户端 ID 和客户端密钥，可以在数据来源 Lambda 函数上找到用于数据来源且作为 AZURE_CLIENT_SECRET 环境变量的密钥 ARN。有关更新 AWS Secrets Manager 中的密钥的更多信息，请参阅修改 AWS Secrets Manager 密钥。

查询数据来源

- 查询 Azure Monitor 时，在多来源查询选项卡中选择数据来源并选择 Azure Monitor 连接器后，即可指定 Azure 订阅以及资源组和资源。然后，您可以选择指标命名空间、指标和聚合，并按维度进行筛选。

Prometheus

创建数据来源

- 您必须提供 Prometheus 端点以及查询 Prometheus 所需的用户名和密码。凭证将存储于 AWS Secrets Manager 内。
- 如果数据来源只能在 VPC 中访问，则连接器必须包含 VPC 配置，如 通过向导连接到预构建数据来源 中所述。如果数据来源要连接到 VPC 以获取凭证，则必须在 VPC 中配置端点。有关更多信息，请参阅使用 AWS Secrets Manager VPC 端点。

更新数据来源配置

- 您可以通过以下步骤手动更新数据来源：
  - 要更新 Prometheus 端点，请将新的端点指定为数据来源 Lambda 函数上的 `PROMETHEUS_API_ENDPOINT` 环境变量。
  - 要更新用于连接到 Prometheus 的用户名和密码，可以在数据来源 Lambda 函数上找到作为 `PROMETHEUS_API_SECRET` 环境变量用于数据来源的密钥的 ARN。有关更新 AWS Secrets Manager 中的密钥的更多信息，请参阅修改 AWS Secrets Manager 密钥。
  - 要更新 VPC 配置，请参阅配置 VPC 访问（控制台）了解更多信息。

查询数据来源

> **Important**
>
> Prometheus 指标类型与 CloudWatch 指标不同，Prometheus 提供的许多指标都是特意累积的。当您查询 Prometheus 指标时，CloudWatch 不会对数据进行任何其他转换：如果您仅指定指标名称或标签，则将显示累积的值。有关更多信息，请参阅 Prometheus 文档中的指标类型。

要将 Prometheus 指标（例如 CloudWatch 指标）数据视为离散值，则需要在运行查询之前对其进行编辑。例如，您可能需要向 Prometheus 指标名称添加对比率函数的调用。有关比率函数和其他 Prometheus 函数的文档，请参阅 Prometheus 文档中的 rate()。

CloudWatch 数据来源连接器不支持多行查询。执行查询，或者使用查询创建警报或控制面板小组件时，每个换行符都会替换为空格。在某些情况下，这可能会导致查询无效。例如，如果您的查询包含单行注释，则该查询无效。如果您尝试利用命令行或基础设施即代码中的多行查询来创建控制面板或警报，API 将拒绝该操作，并显示解析错误。

可用更新的通知

Amazon 可能会不时通知您，推荐您将连接器更新为较新的可用版本，并为您提供操作说明。

创建自定义数据来源连接器。

要将自定义数据来源连接到 CloudWatch，可使用两种方法：

- 首先使用 CloudWatch 提供的示例模板。使用该模板时，您可以使用 JavaScript 或 Python。这些模板包含示例 Lambda 代码，这些代码将有助于您创建 Lambda 函数。然后，您可以修改模板中的 Lambda 函数，以便连接到自定义数据来源。
- 从头开始创建 AWS Lambda 函数，用于实现数据来源连接器、数据查询以及准备供 CloudWatch 使用的时间序列。此函数必须预聚合或合并数据点（如果需要），还必须调整周期和时间戳以与 CloudWatch 兼容。

目录

- 使用模板
- 从头开始创建自定义数据来源
  - 步骤 1：创建函数
    - GetMetricData 事件
    - DescribeGetMetricData 事件
    - CloudWatch 警报的重要注意事项
    - （可选）使用 AWS Secrets Manager 存储凭证
    - （可选）连接到 VPC 中的数据来源
  - 步骤 2：创建 Lambda 权限策略

- 步骤 3：将资源标签附加到 Lambda 函数

使用模板

使用模板可以创建示例 Lambda 函数，并帮助您更快地构建自定义连接器。这些示例函数提供了示例代码，可用于涉及构建自定义连接器的许多常见场景。在使用模板创建连接器后，您可以检查 Lambda 代码，然后对其进行修改以用于连接数据来源。

此外，如果使用该模板，CloudWatch 会负责创建 Lambda 权限策略并将资源标签附加到 Lambda 函数。

使用模板创建连接到自定义数据来源的连接器

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。
3. 选择指标数据来源选项卡。
4. 选择创建数据来源。
5. 选择自定义 - 入门模板单选按钮，然后选择下一步。
6. 输入数据来源的名称。
7. 从列出的模板中选择一个。
8. 选择 Node.js 或 Python。
9. 选择创建数据来源。

   AWS CloudFormation 堆栈完成创建自定义来源之后，您刚刚添加的新自定义来源才会出现。要查看进度，可以选择查看我的 CloudFormation 堆栈的状态。您也可以使用刷新图标来更新此列表。

   新数据来源出现在此列表中时，您就可以在控制台中对其进行测试和修改。
10. （可选）要在控制台中查询来自该来源的测试数据，请按照 创建源自另一个数据来源的指标图表 中的说明进行操作。
11. 按需修改 Lambda 函数。

    a. 在导航窗格中，选择 Settings（设置）。

    b. 选择指标数据来源选项卡。

    c. 针对要修改的来源，选择在 Lambda 控制台中查看。

您现在可以修改函数以访问数据来源。有关更多信息，请参阅 步骤 1：创建函数。

> **Note**
>
> 通过使用该模板，您在编写 Lambda 函数时，无需按照 步骤 2：创建 Lambda 权限策略 或 步骤 3：将资源标签附加到 Lambda 函数 中的说明进行操作。CloudWatch 会执行这些步骤，因为您使用了该模板。

从头开始创建自定义数据来源

按照本节中的步骤创建 Lambda 函数，该函数会将 CloudWatch 连接到数据来源。

步骤 1：创建函数

自定义数据来源连接器必须支持来自 CloudWatch 的 `GetMetricData` 事件。您也可以实施 `DescribeGetMetricData` 活动，以便在 CloudWatch 控制台中向用户提供有关如何使用连接器的文档。`DescribeGetMetricData` 响应还可用于设置 CloudWatch 自定义查询构建器中使用的默认值。

CloudWatch 提供了代码段示例，以帮助您开始使用。有关更多信息，请参阅 https://github.com/aws-samples/cloudwatch-data-source-samples 上的示例存储库。

约束

- 来自 Lambda 的响应必须小于 6 Mb。如果响应超过 6 Mb，则 `GetMetricData` 响应会将 Lambda 函数标记为 `InternalError`，并且不会返回任何数据。
- 如果是用于可视化和控制面板管理，则 Lambda 函数必须在 10 秒内完成执行；如果是用于警报，则 Lambda 函数必须在 4.5 秒内完成执行。如果执行超时，则 `GetMetricData` 响应会将 Lambda 函数标记为 `InternalError`，并且不会返回任何数据。
- Lambda 函数必须使用以秒为单位的纪元时间戳发送其输出。
- 如果 Lambda 函数不对数据进行重采样，而是返回与 CloudWatch 用户请求的开始时间和时段长度不一致的数据，则 CloudWatch 将忽略该数据。任何可视化或警报中都将丢弃多余的数据。未介于开始时间和结束时间之间的任何数据也将被丢弃。

  例如，如果用户要求提供 10:00 到 11:00 之间的数据，周期为 5 分钟，则“10:00:00 至 10:04:59”和“10:05:00 至 10:09:59”为返回数据的有效时间范围。必须返回包含 `10:00`

value1、10:05 value2 等的时间序列。例如，如果函数返回 10:03 valueX，则会被丢弃，因为 10:03 与请求的开始时间和周期不对应。

- CloudWatch 数据来源连接器不支持多行查询。执行查询，或者使用查询创建警报或控制面板小组件时，每个换行符都会替换为空格。在某些情况下，这可能会导致查询无效。

GetMetricData 事件

请求负载

以下是作为输入发送到 Lambda 函数的 GetMetricData 请求负载示例。

```
{
  "EventType": "GetMetricData",
  "GetMetricDataRequest": {
    "StartTime": 1697060700,
    "EndTime": 1697061600,
    "Period": 300,
    "Arguments": ["serviceregistry_external_http_requests{host_cluster!=\"prod\"}"]
  }
}
```

- StartTime：指定最早返回的数据的时间戳。类型为时间戳纪元秒。
- EndTime：指定最新返回的数据的时间戳。类型为时间戳纪元秒。
- Period：每个指标数据聚合所代表的秒数。最短时间为 60 秒。类型为秒。
- Arguments：要传递给 Lambda 指标数学表达式的参数数组。有关传递参数的更多信息，请参阅 如何将参数传递给您的 Lambda 函数。

响应负载

以下是 Lambda 函数返回的 GetMetricData 响应负载的示例。

```
{
   "MetricDataResults": [
      {
         "StatusCode": "Complete",
         "Label": "CPUUtilization",
         "Timestamps": [ 1697060700, 1697061000, 1697061300 ],
         "Values": [ 15000, 14000, 16000 ]
      }
```

```
    ]
}
```

该响应负载将包含 MetricDataResults 字段或 Error 字段，但无法同时包含二者。

MetricDataResults 字段是 MetricDataResult 类型的时间序列字段列表。每个时间序列字段可包括以下字段。

- StatusCode：（可选）Complete 表示已返回请求时间范围内的所有数据点。PartialData 表示返回的数据点不完整。如果省略，则使用默认值 Complete。

  有效值: Complete | InternalError | PartialData | Forbidden
- Messages：可选的消息列表，其中包含所返回的数据的其他相关信息。

  类型：包含 Code 和 Value 字符串的 MessageData 对象数组。
- Label：与数据关联的人类可读标签。

  类型：字符串
- Timestamps：数据点的时间戳，格式为纪元时间。时间戳的数量始终与值的数量相匹配，Timestamps[x] 的值为 Values[x]。

  类型：时间戳数组
- Values：与 Timestamps 对应的指标数据点值。值的数量始终与时间戳的数量相匹配，Timestamps[x] 的值为 Values[x]。

  类型：双倍数组

有关 Error 对象的更多信息，请参阅以下部分。

错误响应格式

您可以选择使用错误响应来提供有关错误的更多信息。当发生验证错误时（例如缺少参数或参数类型错误），我们建议您使用代码验证返回错误。

以下是 Lambda 函数要引发 GetMetricData 验证异常时的响应示例。

```
{
   "Error": {
      "Code": "Validation",
      "Value": "Invalid Prometheus cluster"
```

```
   }
}
```

以下是 Lambda 函数表示由于访问问题而无法返回数据时的响应示例。响应被转换为单个时间序列，其状态代码为 Forbidden。

```
{
   "Error": {
      "Code": "Forbidden",
      "Value": "Unable to access ..."
   }
}
```

以下是 Lambda 函数引发整体 InternalError 异常的示例，该异常被转换为单个时间序列，其状态代码为 InternalError 并具有一条消息。只要错误代码的值不是 Validation 或 Forbidden，CloudWatch 都会认为这属于常规内部错误。

```
{
   "Error": {
      "Code": "PrometheusClusterUnreachable",
      "Value": "Unable to communicate with the cluster"
   }
}
```

DescribeGetMetricData 事件

请求负载

以下是 DescribeGetMetricData 请求负载的示例。

```
{
  "EventType": "DescribeGetMetricData"
}
```

响应负载

以下是 DescribeGetMetricData 响应负载的示例。

```
{
    "Description": "Data source connector",
    "ArgumentDefaults": [{
        Value: "default value"
```

```
    }]
}
```

- Description：关于如何使用数据来源连接器的描述。此描述将显示在 CloudWatch 控制台中。支持 Markdown。

  类型：字符串
- ArgumentDefaults：预填充自定义数据来源构建器时使用的参数默认值的可选数组。

  如果返回 `[{ Value: "default value 1"}, { Value: 10}]`，则 CloudWatch 控制台中的查询构建器会显示两个输入，第一个是“default value 1”，第二个是“10”。

  如果未提供 `ArgumentDefaults`，则显示默认类型为 `String` 的一个输入。

  类型：包含值和类型的对象数组。
- Error：（可选）错误字段可包含在任何响应中。您可以在 GetMetricData 事件 中查看示例。

CloudWatch 警报的重要注意事项

如果要使用数据来源来设置 CloudWatch 警报，则应将其设置为每分钟向 CloudWatch 报告带有时间戳的数据。有关为源自已连接的数据来源的指标创建警报的更多信息和其他注意事项，请参阅 基于连接的数据来源创建警报。

（可选）使用 AWS Secrets Manager 存储凭证

如果您的 Lambda 函数需要使用凭证来访问数据来源，我们建议使用 AWS Secrets Manager 来存储这些凭证，而不是将其硬编码到您的 Lambda 函数中。有关结合使用 AWS Secrets Manager 和 Lambda 函数的更多信息，请参阅在 AWS Lambda 函数中使用 AWS Secrets Manager 密钥。

（可选）连接到 VPC 中的数据来源

如果数据来源位于由 Amazon Virtual Private Cloud 管理的 VPC 中，则必须配置您的 Lambda 函数才能对其进行访问。有关更多信息，请参阅将出站联网连接到 VPC 中的资源。

您可能还需要配置 VPC 服务端点才能访问 AWS Secrets Manager 等服务。有关更多信息，请参阅使用接口 VPC 端点访问 AWS 服务。

步骤 2：创建 Lambda 权限策略

您必须创建策略语句，授予 CloudWatch 使用您创建的 Lambda 函数的权限。您可以使用 AWS CLI 或 Lambda 控制台创建该策略语句。

使用 AWS CLI 创建策略语句

- 输入以下 命令。将 *123456789012* 替换为您的账户 ID，将 *my-data-source-function* 替换为您的 Lambda 函数的名称，然后将 *MyDataSource-DataSourcePermission1234* 替换为任意唯一值。

```
aws lambda add-permission --function-name my-data-source-function --statement-
id MyDataSource-DataSourcePermission1234 --action lambda:InvokeFunction --principal
 lambda.datasource.cloudwatch.amazonaws.com --source-account 123456789012
```

步骤 3：将资源标签附加到 Lambda 函数

CloudWatch 控制台通过使用标签来确定哪些 Lambda 函数是数据来源连接器。使用其中一个向导创建数据来源时，配置该标签的 AWS CloudFormation 堆栈会自动应用该标签。当您自己创建数据来源时，可以将以下标签用于您的 Lambda 函数。这样一来，当您查询指标时，您的连接器就会显示在 CloudWatch 控制台的数据来源下拉列表中。

- 具有键 `cloudwatch:datasource` 和值 `custom` 的标签。

使用您的自定义数据来源

创建数据来源后，您可以使用它查询源自该来源的数据，以对其进行可视化并设置警报。如果您使用模板创建了自定义数据来源连接器，或者添加了 步骤 3：将资源标签附加到 Lambda 函数 中列出的标签，则可以按照 创建源自另一个数据来源的指标图表 中的步骤进行查询。

您也可以使用指标数学函数 LAMBDA 对其进行查询，如下节所述。

有关对数据来源的指标创建警报的更多信息，请参阅 基于连接的数据来源创建警报。

如何将参数传递给您的 Lambda 函数

要向自定义数据来源传递参数，建议您在查询数据来源时使用 CloudWatch 控制台中的查询构建器。

您还可以通过使用 CloudWatch 指标数学中的新 LAMBDA 表达式，来使用 Lambda 函数检索数据来源中的数据。

```
LAMBDA("LambdaFunctionName" [, optional-arg]*)
```

`optional-arg` 最多有 20 个字符串、数字或布尔值。例如，`param`、`3.14` 或 `true`。

> **Note**
>
> CloudWatch 数据来源连接器不支持多行字符串。执行查询，或者使用查询创建警报或控制面板小组件时，每个换行符都会替换为空格。在某些情况下，这可能会导致查询无效。

使用 LAMBDA 指标数学函数时，可以提供函数名称 ("MyFunction")。在资源策略允许的情况下，您还可以使用特定版本的函数 ("MyFunction:22") 或使用 Lambda 函数别名 ("MyFunction:MyAlias")。您无法使用 *

以下是调用 LAMBDA 函数的一些示例。

```
LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query")
```

```
LAMBDA("MyCustomDataSource", true, "fuzzy", 99.9)
```

LAMBDA 指标数学函数会返回一个时间序列列表，该列表可以返回到请求者或与其他指标数学函数结合使用。以下是 LAMBDA 与其他指标数学函数结合使用的示例。

```
FILL(LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query"), 0)
```

删除数据来源连接器

要删除数据来源连接器，请按照本节中的说明操作。

删除数据来源连接器

1. 通过 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择 Settings（设置）。
3. 选择指标数据来源选项卡。
4. 在要删除的数据来源的行中，选择在 CloudFormation 中管理。

   您会进入 AWS CloudFormation 控制台。
5. 在带有数据来源名称的部分中，选择删除。
6. 在确认弹出窗口中，选择删除。

使用 CloudWatch 代理收集指标、日志和跟踪信息

您可以通过统一 CloudWatch 代理执行以下操作：

- 跨操作系统从 Amazon EC2 实例中收集内部系统级指标。除了 EC2 实例的指标之外，这些指标还可以包括来宾中的指标。可以收集的其他指标列在CloudWatch 代理收集的指标中。
- 从本地服务器中收集系统级别指标。这些服务器可能包括混合环境中的服务器以及不是由 AWS 管理的服务器。
- 使用 `StatsD` 和 `collectd` 协议从应用程序或服务中检索自定义指标。`StatsD` 在 Linux 服务器和运行 Windows Server 的服务器上都受支持。`collectd` 仅在 Linux 服务器上受支持。
- 从运行 Linux 或 Windows Server 的 Amazon EC2 实例和本地部署服务器收集日志。

> **Note**
>
> CloudWatch 代理不支持从 FIFO 管道收集日志。

- 版本 1.300031.0 及更高版本可用于启用 CloudWatch Application Signals。有关更多信息，请参阅 Application Signals。
- 版本 1.300025.0 及更高版本可以从 OpenTelemetry 或 X-Ray 客户端开发工具包中收集跟踪数据，并将这些跟踪数据发送到 X-Ray。

  使用 CloudWatch 代理，您可以轻松收集跟踪，而无需运行单独的跟踪收集进程守护程序，这有助于减少运行和管理的代理数量。

您可以在 CloudWatch 中存储和查看使用 CloudWatch 代理收集的指标，就像任何其他 CloudWatch 指标一样。CloudWatch 代理收集的指标的默认命名空间为 `CWAgent`，不过您可以在配置该代理时指定其他命名空间。

由统一 CloudWatch 代理收集的日志在 Amazon CloudWatch Logs中处理和存储，就像较旧的 CloudWatch Logs 代理收集的日志一样。有关 CloudWatch Logs 定价的信息，请参阅 Amazon CloudWatch 定价。

CloudWatch 代理收集的指标按自定义指标计费。有关 CloudWatch 指标定价的信息，请参阅 Amazon CloudWatch 定价。

CloudWatch 代理是 MIT 许可证下的开源代理，并且托管在 GitHub 上。如果您想要构建、自定义或贡献 CloudWatch 代理，请参阅 GitHub 存储库以获取最新说明。如果您认为您发现了潜在的安全问题，

请勿将其发布到 GitHub 或任何公共论坛上。请按照漏洞报告中的说明进行操作，或者直接向 AWS 安全发送电子邮件。

本节中的步骤介绍如何在 Amazon EC2 实例和本地部署服务器上安装统一 CloudWatch 代理。有关 CloudWatch 代理能够收集的指标的更多信息，请参阅 CloudWatch 代理收集的指标。

支持的操作系统

在以下操作系统的 x86-64 架构上支持 CloudWatch 代理。此处列出的每个主要版本的所有次要版本更新也支持该代理。

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server 版本 23.10、22.04、20.04、18.04、16.04 和 14.04
- CentOS 版本 9、8 和 7
- Red Hat Enterprise Linux（RHEL）版本 9、8 和 7
- Debian 版本 12、11 和 10
- SUSE Linux Enterprise Server（SLES）版本 15 和 12
- Oracle Linux 版本 9、8 和 7
- AlmaLinux 版本 9 和 8
- Rocky Linux 版本 9 和 8
- 以下 macOS 计算机：EC2 M1 Mac1 实例以及运行 macOS 14（Sonoma）、macOS 13（Ventura）和 macOS 12（Monterey）的计算机
- 64 位版本的 Windows Server 2022、Windows Server 2019 和 Windows Server 2016
- 64 位 Windows 10

在以下操作系统的 ARM64 架构上支持该代理。此处列出的每个主要版本的所有次要版本更新也支持该代理。

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server 版本 23.10、22.04、20.04、18.04 和 16.04
- CentOS 版本 9 和 8
- Red Hat Enterprise Linux（RHEL）版本 9、8 和 7
- Debian 版本 12、11 和 10

- SUSE Linux Enterprise Server 15
- 以下 macOS 计算机：macOS 14（Sonoma）、macOS 13（Ventura）和 macOS 12（Monterey）

安装过程概览

您可以使用命令行手动下载并安装 CloudWatch 代理，也可以将其与 SSM 集成。使用这两种方法之一安装 CloudWatch 代理的一般流程如下所示：

1. 创建使代理能够从服务器中收集指标且（可选）与 AWS Systems Manager 集成的 IAM 角色或用户。
2. 下载代理软件包。
3. 修改 CloudWatch 代理配置文件并指定要收集的指标。
4. 在服务器上安装并启动代理。当您在 EC2 实例上安装代理时，将会附加在步骤 1 中创建的 IAM 角色。当您在本地部署服务器上安装代理时，您会指定一个命名的配置文件，其中包含您在步骤 1 中创建的 IAM 用户的凭证。

内容

- 安装 CloudWatch 代理
- 使用 AWS CloudFormation 在新实例上安装 CloudWatch 代理
- CloudWatch 代理凭证首选项
- 验证 CloudWatch 代理软件包的签名
- 创建 CloudWatch 代理配置文件
- 使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理
- CloudWatch 代理收集的指标
- CloudWatch 代理的常见场景
- CloudWatch 代理故障排除

安装 CloudWatch 代理

CloudWatch 代理在 Amazon Linux 2023 和 Amazon Linux 2 中，可作为软件包使用。如果您使用其中一个操作系统，则可以通过输入以下命令来安装该软件包。您还必须确保附加到实例的 IAM 角色附加了 CloudWatchAgentServerPolicy。有关更多信息，请参阅 在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。

```
sudo yum install amazon-cloudwatch-agent
```

在所有支持的操作系统（包括 Linux 和 Windows Server）中，您可以使用命令行以及 Amazon S3 下载链接、使用 Amazon EC2 Systems Manager 或使用 AWS CloudFormation 模板来下载并安装 CloudWatch 代理。有关详细信息，请参阅下面几节：

内容

- 使用命令行安装 CloudWatch 代理
- 使用 AWS Systems Manager 安装 CloudWatch 代理
- 在本地服务器上安装 CloudWatch 代理

使用命令行安装 CloudWatch 代理

使用以下主题来下载、配置和安装 CloudWatch 代理软件包。

主题

- 使用命令行下载和配置 CloudWatch 代理
- 创建 IAM 角色和用户以用于 CloudWatch 代理
- 在服务器上安装和运行 CloudWatch 代理

使用命令行下载和配置 CloudWatch 代理

使用以下步骤来下载 CloudWatch 代理软件包，创建 IAM 角色或用户，以及（可选）根据需要修改常见配置文件。

下载 CloudWatch 代理软件包

> **Note**
>
> 要下载 CloudWatch 代理，您的连接必须使用 TLS 1.2 或更高版本。

CloudWatch 代理在 Amazon Linux 2023 和 Amazon Linux 2 中，可作为软件包使用。如果您使用此操作系统，则可以通过输入以下命令来安装该软件包。您还必须确保附加到实例的 IAM 角色附加了 CloudWatchAgentServerPolicy。有关更多信息，请参阅 创建 IAM 角色和用户以用于 CloudWatch 代理。

```
sudo yum install amazon-cloudwatch-agent
```

在所有支持的操作系统中，您可以使用命令行下载并安装 CloudWatch 代理。

对于每个下载链接，有一个常规链接以及每个区域的链接。例如，对于 Amazon Linux 2023 和 Amazon Linux 2 以及 x86-64 架构，三个有效的下载链接如下：

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

您也可以下载有关代理的最新更改的 README 文件，以及指示可供下载的版本号的文件。这些文件位于以下位置：

- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES` 或 `https://amazoncloudwatch-agent-`*`region`*`.s3.`*`region`*`.amazonaws.com/info/latest/RELEASE_NOTES`
- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION` 或 `https://amazoncloudwatch-agent-`*`region`*`.s3.`*`region`*`.amazonaws.com/info/latest/CWAGENT_VERSION`

架构	平台	下载链接	签名文件链接
x86-64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
		https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/la	https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/la

架构	平台	下载链接	签名文件链接
		test/amazon-cloudwatch-agent.rpm	test/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

架构	平台	下载链接	签名文件链接
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

架构	平台	下载链接	签名文件链接
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig

架构	平台	下载链接	签名文件链接
ARM64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

架构	平台	下载链接	签名文件链接
ARM64	MacOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig

使用命令行下载和安装 CloudWatch 代理软件包

1. 下载 CloudWatch 代理。

   在 Linux 服务器上，输入以下命令。对于 *download-link*，请使用上表中的相应下载链接。

   ```
   wget download-link
   ```

   在运行 Windows Server 的服务器上，下载以下文件：

   ```
   https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
   ```

2. 在下载该软件包后，您可以选择验证软件包签名。有关更多信息，请参阅 验证 CloudWatch 代理软件包的签名。
3. 安装 软件包。如果已在 Linux 服务器上下载 RPM 程序包，请更改为包含程序包的目录，并输入以下内容：

   ```
   sudo rpm -U ./amazon-cloudwatch-agent.rpm
   ```

   如果已在 Linux 服务器上下载 DEB 程序包，请更改为包含程序包的目录，并输入以下内容：

   ```
   sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
   ```

如果已运行在 Windows Server 的服务器上下载 MSI 程序包，请更改为包含程序包的目录，并输入以下内容：

```
msiexec /i amazon-cloudwatch-agent.msi
```

此命令还可在 PowerShell 中发挥作用。有关 MSI 命令选项的更多信息，请参阅 Microsoft Windows 文档中的命令行选项。

如果已在 macOS 服务器上下载 PKG 程序包，请更改为包含程序包的目录，并输入以下内容：

```
sudo installer -pkg ./amazon-cloudwatch-agent.pkg -target /
```

创建和修改代理配置文件

在下载 CloudWatch 代理后，您必须先创建配置文件，然后再在任何服务器上启动该代理。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。

创建 IAM 角色和用户以用于 CloudWatch 代理

要访问 AWS 资源，需要具有相应的权限。您创建 IAM 角色和/或 IAM 用户，以授予 CloudWatch 代理将指标写入 CloudWatch 所需的权限。如果您打算在 Amazon EC2 实例上使用代理，则必须创建 IAM 角色。如果您打算在本地部署服务器上使用代理，则必须创建 IAM 用户。

> **Note**
>
> 我们最近修改了以下过程，具体来说就是使用由 Amazon 创建的新 `CloudWatchAgentServerPolicy` 和 `CloudWatchAgentAdminPolicy` 策略，而不需要客户自行创建这些策略。将文件写入 Parameter Store 和从 Parameter Store 中下载文件时，由 Amazon 创建的策略仅支持名称以 `AmazonCloudWatch-` 开头的文件。如果您的 CloudWatch 代理配置文件的文件名不以 `AmazonCloudWatch-` 开头，则这些策略不能用于将文件写入 Parameter Store 或从 Parameter Store 下载文件。

如果您打算在 Amazon EC2 实例上运行 CloudWatch 代理，请使用以下步骤创建必要的 IAM 角色。该角色提供从实例中读取信息并将其写入到 CloudWatch 的权限。

创建在 EC2 实例上运行 CloudWatch 代理所需的 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择 Roles (角色)，然后选择 Create role (创建角色)。
3. 确保在 Trusted entity type（受信任实体的类型）下选择了 AWS service（服务）。
4. 对于 Use case（使用案例），在 Common use cases（常见使用案例）下选择 EC2，
5. 选择下一步。
6. 在策略列表中，选中 CloudWatchAgentServerPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
7. （可选）如果代理要将跟踪数据发送到 X-Ray，您还需要为角色提供 AWSXRayDaemonWriteAccess 策略。为此，请在列表中找到该策略，然后选中它旁边的复选框。
8. 选择下一步。
9. 在 Role name (角色名称) 中，输入角色的名称，例如 *CloudWatchAgentServerRole*。（可选）为其指定说明。然后选择创建角色。

   将立即创建该角色。
10. （可选）如果代理将向 CloudWatch Logs 发送日志，并且您想要代理能够为这些日志组设置保留策略，则需要将 `logs:PutRetentionPolicy` 权限添加到角色。有关更多信息，请参阅 允许 CloudWatch 代理设置日志保留策略。

如果您打算在本地部署服务器上运行 CloudWatch 代理，请使用以下步骤创建必要的 IAM 用户。

> **Warning**
>
> 此场景需要 IAM 用户具有编程访问权限和长期凭证，这会带来安全风险。为帮助减轻这种风险，我们建议仅向这些用户提供执行任务所需的权限，并在不再需要这些用户时将其移除。必要时可以更新访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 更新访问密钥。

创建 CloudWatch 代理在本地部署服务器上运行所需的 IAM 用户

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择 Users（用户），然后选择 Add user（添加用户）。

3. 为新用户输入用户名。
4. 选择 Access key - Programmatic access（访问密钥 - 编程访问），然后选择 Next: Permissions（下一步: 权限）。
5. 选择直接附上现有策略。
6. 在策略列表中，选中 CloudWatchAgentServerPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
7. （可选）如果代理要跟踪到 X-Ray，您还需要为角色提供 AWSXRayDaemonWriteAccess 策略。为此，请在列表中找到该策略，然后选中它旁边的复选框。
8. 选择下一步：标签。
9. 可以选择为新 IAM 用户创建标签，然后选择 Next: Review（下一步: 审核）。
10. 确认列出了正确的策略，然后选择 Create user（创建用户）。
11. 在新用户的名称旁边，选择显示。将访问密钥和私有密钥复制到一个文件以便在安装该代理时使用。选择关闭。

允许 CloudWatch 代理设置日志保留策略

您可以配置 CloudWatch 代理，以便为接收其发送的日志事件的日志组设置保留策略。如果您这样做，您必须向代理使用的 IAM 角色或用户授予 `logs:PutRetentionPolicy`。代理使用 IAM 角色在 Amazon EC2 实例上运行，使用本地部署服务器的 IAM 用户。

向 CloudWatch 代理的 IAM 角色授予设置日志保留策略的权限

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在搜索框中，键入 CloudWatch 代理的 IAM 角色名称的开头部分。您在创建角色时选择了此名称。其可能被命名为 `CloudWatchAgentServerRole`。

   当您看到角色时，选择角色的名称。
4. 在 Permissions（权限）选项卡中，依次选择 Add permissions（添加权限）和 Create inline policy（创建内联策略）。
5. 选择 JSON 选项卡并将以下策略复制到框中，替换框中默认的 JSON:

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. 选择查看策略。
7. 对于 Name（名称），输入 **CloudWatchAgentPutLogsRetention** 或类似内容，然后选择 Create policy（创建策略）。

向 CloudWatch 代理的 IAM 用户授予设置日志保留策略的权限

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧导航窗格中，选择 用户。
3. 在搜索框中，键入 CloudWatch 代理的 IAM 用户名称的开头部分。您在创建用户时选择了此名称。

   当您看到用户时，选择用户的名称。
4. 在权限选项卡中，选择 添加内联策略。
5. 选择 JSON 选项卡并将以下策略复制到框中，替换框中默认的 JSON:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. 选择查看策略。
7. 对于 Name（名称），输入 **CloudWatchAgentPutLogsRetention** 或类似内容，然后选择 Create policy（创建策略）。

在服务器上安装和运行 CloudWatch 代理

在创建所需的代理配置文件并创建 IAM 角色或 IAM 用户后，使用该配置执行以下步骤在服务器上安装和运行代理。首先，将 IAM 角色或 IAM 用户附加到将要运行代理的服务器上。然后，在该服务器上，下载代理软件包并使用您创建的代理配置文件启动它。

使用 S3 下载链接下载 CloudWatch 代理软件包

> **Note**
>
> 要下载 CloudWatch 代理，您的连接必须使用 TLS 1.2 或更高版本。

您需要在将要运行代理的每个服务器上安装代理。

Amazon Linux AMI

CloudWatch 代理在 Amazon Linux 2023 和 Amazon Linux 2 中，可作为软件包使用。如果您使用此操作系统，则可以通过输入以下命令来安装该软件包。您还必须确保附加到实例的 IAM 角色附加了 CloudWatchAgentServerPolicy。有关更多信息，请参阅 在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。

```
sudo yum install amazon-cloudwatch-agent
```

所有操作系统

在所有支持的操作系统中，您可以使用命令行以及以下步骤中所列的 Amazon S3 下载链接来下载并安装 CloudWatch 代理。

对于每个下载链接，有一个常规链接以及每个区域的链接。例如，对于 Amazon Linux 2023 和 Amazon Linux 2 以及 x86-64 架构，三个有效的下载链接如下：

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

架构	平台	下载链接	签名文件链接
x86-64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/	https://amazoncloudwatch-agent.s3.amazonaws.com/

架构	平台	下载链接	签名文件链接
		suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig

架构	平台	下载链接	签名文件链接
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig

架构	平台	下载链接	签名文件链接
ARM64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

架构	平台	下载链接	签名文件链接
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

使用命令行在 Amazon EC2 实例上安装 CloudWatch 代理

1. 下载 CloudWatch 代理。对于 Linux 服务器，请输入以下命令。对于 *download-link*，请使用上表中的相应下载链接。

   ```
   wget download-link
   ```

   对于运行 Windows Server 的服务器，请下载以下文件：

   ```
   https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
   ```

2. 在下载该软件包后，您可以选择验证软件包签名。有关更多信息，请参阅 验证 CloudWatch 代理软件包的签名。

3. 安装 软件包。如果已在 Linux 服务器上下载 RPM 程序包，请更改为包含程序包的目录，并输入以下内容：

   ```
   sudo rpm -U ./amazon-cloudwatch-agent.rpm
   ```

   如果已在 Linux 服务器上下载 DEB 程序包，请更改为包含程序包的目录，并输入以下内容：

   ```
   sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
   ```

   如果已运行在 Windows Server 的服务器上下载 MSI 程序包，请更改为包含程序包的目录，并输入以下内容：

```
msiexec /i amazon-cloudwatch-agent.msi
```

此命令还可在 PowerShell 中发挥作用。有关 MSI 命令选项的更多信息，请参阅 Microsoft Windows 文档中的命令行选项。

（在 EC2 实例上安装）附加 IAM 角色

要使 CloudWatch 代理能够从实例发送数据，您必须将 IAM 角色附加到实例。要附加的角色是 CloudWatchAgentServerRole。您之前应该已经创建了此角色。有关更多信息，请参阅创建 IAM 角色和用户以用于 CloudWatch 代理。

有关将 IAM 角色附加到实例的更多信息，请参阅《Amazon EC2 用户指南》中的将 IAM 角色附加到实例。

（安装在本地部署服务器上）指定 IAM 凭证和 AWS 区域

要使 CloudWatch 代理能够从本地部署服务器发送数据，您必须指定先前创建的 IAM 用户的访问密钥和私有密钥。有关创建此用户的更多信息，请参阅创建 IAM 角色和用户以用于 CloudWatch 代理。

您还必须使用 AWS 配置文件的 `[AmazonCloudWatchAgent]` 部分中的 `region` 字段指定要将指标发送到的 AWS 区域，如以下示例中所示。

```
[profile AmazonCloudWatchAgent]
region = us-west-1
```

下面是使用 `aws configure` 命令为 CloudWatch 代理创建命名配置文件的示例。该示例假设您使用默认配置文件名称 `AmazonCloudWatchAgent`。

为 CloudWatch 代理创建 AmazonCloudWatchAgent 配置文件

1. 如果您还没有这样做，请在服务器上安装 AWS Command Line Interface。有关更多信息，请参阅安装 AWS CLI。
2. 在 Linux 服务器上，输入以下命令并按照提示进行操作：

```
sudo aws configure --profile AmazonCloudWatchAgent
```

在 Windows Server 上，以管理员身份打开 PowerShell，输入以下命令并按照提示进行操作。

```
aws configure --profile AmazonCloudWatchAgent
```

验证互联网访问权限

您的 Amazon EC2 实例必须具有出站互联网访问权限，才能将数据发送到 CloudWatch 或 CloudWatch Logs。有关如何配置互联网访问权限的更多信息，请参阅 Amazon VPC 用户指南中的互联网网关。

要在您的代理上配置的终端节点和端口如下所示：

- 如果要使用代理收集指标，则必须将相应区域的 CloudWatch 端点加入白名单。这些端点在 Amazon CloudWatch 端点和配额中列出。
- 如果要使用代理收集日志，则必须将相应区域的 CloudWatch Logs 端点加入白名单。这些端点在 Amazon CloudWatch Logs 端点和配额中列出。
- 如果使用 Systems Manager 安装代理或使用 Parameter Store 存储配置文件，您必须将相应区域的 Systems Manager 端点加入白名单。这些端点在 AWS Systems Manager 端点和配额中列出。

（可选）修改代理的通用配置或区域信息

CloudWatch 代理包含一个名为 `common-config.toml` 的配置文件。您可以（可选）使用该文件指定代理和区域信息。

在运行 Linux 的服务器上，该文件位于 `/opt/aws/amazon-cloudwatch-agent/etc` 目录中。在运行 Windows Server 的服务器上，该文件位于 `C:\ProgramData\Amazon\AmazonCloudWatchAgent` 目录中。

> **Note**
>
> 建议您在本地模式下运行 CloudWatch 代理时使用 `common-config.toml` 文件来提供共享配置和凭证，当您在 Amazon EC2 上运行并且想要重复使用现有的共享凭证配置文件和文件时，该文件也非常有用。通过 `common-config.toml` 启用该文件还有一个额外的好处，那就是如果您的共享凭证文件在到期后使用续订的凭证进行轮换，则代理无需重启即可自动选取新凭证。

默认 `common-config.toml` 如下所示。

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for the on-premises case by
 default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

最初将注释所有行。要设置凭证配置文件或代理设置，请从该行中删除 # 并指定一个值。您可以手动编辑该文件，或者使用 Systems Manager 中的 RunShellScript Run Command 执行该操作：

- shared_credential_profile – 对于本地部署服务器，此行指定 IAM 用户凭证配置文件，以用于将数据发送到 CloudWatch。如果您将此行注释掉，则会使用 AmazonCloudWatchAgent。有关创建此配置文件的更多信息，请参阅（安装在本地部署服务器上）指定 IAM 凭证和 AWS 区域。

  在 EC2 实例上，您可以使用此行让 CloudWatch 代理将数据从该实例发送到不同 AWS 区域中的 CloudWatch。要执行此操作，请指定一个包含 region 字段的命名配置文件，该字段指定要发送到的区域的名称。

  如果指定 shared_credential_profile，您还必须从 [credentials] 行开头删除 #。

- shared_credential_file – 要让代理在位于默认路径以外的路径中的文件中查找凭证，请在此处指定完整的路径和文件名。在 Linux 上，默认路径为 /root/.aws；在 Windows Server 上，默认路径为 C:\\Users\\Administrator\\.aws。

  下面的第一个示例显示对 Linux 服务器有效的 shared_credential_file 行的语法，第二个示例对 Windows Server 有效。在 Windows Server 上，您必须转义 \ 字符。

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

如果指定 shared_credential_file，您还必须从 [credentials] 行开头删除 #。

- 代理设置 – 如果您的服务器使用 HTTP 或 HTTPS 代理联系 AWS 服务，请在 http_proxy 和 https_proxy 字段中指定这些代理。如果应从代理中排除某些 URL，请在 no_proxy 字段中指定这些 URL 并以逗号分隔。

使用命令行启动 CloudWatch 代理

可以执行以下步骤以使用命令行在服务器上启动 CloudWatch 代理。

使用命令行在服务器上启动 CloudWatch 代理

1. 将要使用的代理配置文件复制到要运行代理的服务器。记下将其复制到的路径名。
2. 此命令中，-a fetch-config 会使代理加载最新版本的 CloudWatch 代理配置文件，-s 则会启动该代理。

   输入下列命令之一。将 *configuration-file-path* 替换为指向代理配置文件的路径。如果您使用向导创建此文件，则此文件命名为 config.json；如果您手动创建该文件，则该文件可能命名为 amazon-cloudwatch-agent.json。

   在运行 Linux 的 EC2 实例上，输入以下命令。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
   ```

   在运行 Linux 的本地服务器上，输入以下内容：

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
   ```

   在运行 Windows Server 的 EC2 实例上，从 PowerShell 控制台中输入以下内容：

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m ec2 -s -c file:configuration-file-path
```

在运行 Windows Server 的本地服务器上，从 PowerShell 控制台中输入以下内容：

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c file:configuration-file-path
```

使用 AWS Systems Manager 安装 CloudWatch 代理

可以按照以下主题使用 AWS Systems Manager 安装和运行 CloudWatch 代理。

使用 Systems Manager 更轻松地在 Amazon EC2 实例集上安装 CloudWatch 代理。您可以将代理下载到一台服务器上，然后创建要用于实例集中所有服务器的 CloudWatch 代理配置文件。然后，您可以使用 Systems Manager，通过创建的配置文件将代理安装到其他服务器上。

主题

- 创建 IAM 角色和用户以用于 CloudWatch 代理
- 使用 SSM 下载、配置和运行 CloudWatch 代理

创建 IAM 角色和用户以用于 CloudWatch 代理

要访问 AWS 资源，需要具有相应的权限。您可以创建包含所需权限的 IAM 角色和用户，以使 CloudWatch 代理将指标写入到 CloudWatch 中，以及使 CloudWatch 代理与 Amazon EC2 和 AWS Systems Manager 进行通信。您在 Amazon EC2 实例上使用 IAM 角色，在本地部署服务器上使用 IAM 用户。

一个角色或用户能让 CloudWatch 代理安装到服务器上，并将指标发送到 CloudWatch。需要使用另一个角色或用户以在 Systems Manager Parameter Store 中存储 CloudWatch 代理配置。Parameter Store 允许多个服务器使用一个 CloudWatch 代理配置。

写入 Parameter Store 是一项广泛而强大的权限。您应该仅在需要时使用它，并且不应将其附加到部署中的多个实例。如果在 Parameter Store 中存储 CloudWatch 代理配置，我们建议您执行以下操作：

- 设置一个实例以在其中执行该配置。
- 使用仅在该实例上有权限写入到 Parameter Store 的 IAM 角色。

- 使用仅在处理并保存 CloudWatch 代理配置文件时，有权限写入到 Parameter Store 的 IAM 角色。

> **Note**
>
> 我们最近修改了以下过程，具体来说就是使用由 Amazon 创建的新 `CloudWatchAgentServerPolicy` 和 `CloudWatchAgentAdminPolicy` 策略，而不需要客户自行创建这些策略。要使用这些策略将代理配置文件写入 Parameter Store，然后从 Parameter Store 中下载该文件，您的代理配置文件的名称必须以 `AmazonCloudWatch-` 开头。如果您的 CloudWatch 代理配置文件的文件名不以 `AmazonCloudWatch-` 开头，则这些策略不能用于将文件写入 Parameter Store 或从 Parameter Store 下载文件。

在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色

第一个过程会创建一个 IAM 角色，您必须将其附加到运行 CloudWatch 代理的每个 Amazon EC2 实例。该角色提供从实例中读取信息并将其写入到 CloudWatch 的权限。

第二个过程会创建一个 IAM 角色，您必须将其附加到用于创建 CloudWatch 代理配置文件的 Amazon EC2 实例。如果要将该文件存储在 Systems Manager Parameter Store 中，以便其他服务器可以使用该文件，则需要执行该步骤。除了从实例中读取信息并将其写入到 CloudWatch 的权限以外，该角色还提供写入到 Parameter Store 的权限。该角色包含足以运行 CloudWatch 代理以及写入到 Parameter Store 的权限。

> **Note**
>
> Parameter Store 支持标准层和高级层中的参数。这些参数层与 CloudWatch 代理预定义指标集提供的“基本”、“标准”和“高级”详细信息级别无关。

创建每个服务器运行 CloudWatch 代理所需的 IAM 角色

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在 选择受信任实体的类型 下，选择 AWS 服务。
4. 在 Common use cases（常见使用案例）下，选择 EC2，然后选择 Next: Permissions（下一步：权限）。

5. 在策略列表中，选中 CloudWatchAgentServerPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
6. 要使用 Systems Manager 安装或配置 CloudWatch 代理，请选中 AmazonSSMManagedInstanceCore 旁边的框。此 AWS 托管策略使实例能够使用 Systems Manager 服务核心功能。如有必要，请使用搜索框查找该策略。如果要仅通过命令行启动和配置代理，则不需要此策略。
7. 选择 下一步：标签。
8. （可选）添加一个或多个标签键值对，以组织、跟踪或控制此角色的访问，然后选择 Next: Review (下一步：审核)。
9. 对于角色名称，请输入新角色的名称，如 **`CloudWatchAgentServerRole`**或所需其他名称。
10. （可选）对于角色描述，请输入描述。
11. 确认 CloudWatchAgentServerPolicy 和（可选）AmazonSSMManagedInstanceCore 是否显示在 Policies（策略）旁边。
12. 选择 Create role（创建角色）。

    将立即创建该角色。

以下过程创建也可以写入到 Parameter Store 的 IAM 角色。您可以使用该角色在 Parameter Store 中存储代理配置文件，以便其他服务器检索该文件。

写入到 Parameter Store 的权限提供了广泛的访问。此角色不应附加到您的所有服务器，只有管理员才能使用它。在创建代理配置文件并将其复制到 Parameter Store 后，应将该角色从实例中分离并改用 `CloudWatchAgentServerRole`。

为管理员创建 IAM 角色以写入到 Parameter Store

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在 选择受信任实体的类型 下，选择 AWS 服务。
4. 在紧靠选择将使用此角色的服务下面，选择 EC2，然后选择下一步: 权限。
5. 在策略列表中，选中 CloudWatchAgentAdminPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
6. 要使用 Systems Manager 安装或配置 CloudWatch 代理，请选中 AmazonSSMManagedInstanceCore 旁边的框。此 AWS 托管策略使实例能够使用 Systems

Manager 服务核心功能。如有必要，请使用搜索框查找该策略。如果要仅通过命令行启动和配置代理，则不需要此策略。

7. 选择 下一步：标签。
8. （可选）添加一个或多个标签键值对，以组织、跟踪或控制此角色的访问，然后选择 Next: Review (下一步：审核)。
9. 对于角色名称，请输入新角色的名称，如 **CloudWatchAgentAdminRole**或所需其他名称。
10. （可选）对于角色描述，请输入描述。
11. 确认 CloudWatchAgentAdminPolicy 和（可选）AmazonSSMManagedInstanceCore 是否显示在 Policies（策略）旁边。
12. 选择 Create role（创建角色）。

    将立即创建该角色。

创建 IAM 用户以用于本地部署服务器上的 CloudWatch 代理

第一个过程会创建运行 CloudWatch 代理所需的 IAM 用户。该用户提供将数据发送到 CloudWatch 的权限。

第二个过程会创建一个 IAM 用户，您可以在创建 CloudWatch 代理配置文件时使用该用户。可以使用此过程将该文件存储在 Systems Manager Parameter Store 中，以便其他服务器使用该文件。除了将数据写入到 CloudWatch 的权限以外，该用户还提供了写入到 Parameter Store 的权限。

> **Note**
>
> Parameter Store 支持标准层和高级层中的参数。这些参数层与 CloudWatch 代理预定义指标集提供的“基本”、“标准”和“高级”详细信息级别无关。

创建 CloudWatch 代理将数据写入 CloudWatch 所需的 IAM 用户

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择 Users，然后选择 Add user。
3. 为新用户输入用户名。
4. 对于访问类型，请选择编程访问，然后选择下一步：权限。
5. 对于设置权限，请选择直接附加现有策略。

6. 在策略列表中，选中 CloudWatchAgentServerPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
7. 要使用 Systems Manager 安装或配置 CloudWatch 代理，请选中 AmazonSSMManagedInstanceCore 旁边的框。此 AWS 托管策略使实例能够使用 Systems Manager 服务核心功能。（如有必要，请使用搜索框查找策略。如果仅通过命令行启动和配置该代理，则不需要使用该策略。）
8. 选择 下一步：标签。
9. （可选）添加一个或多个标签键值对，以组织、跟踪或控制此角色的访问，然后选择 Next: Review (下一步：审核)。
10. 确认列出了正确的策略，然后选择创建用户。
11. 在新用户的行中，选择显示。将访问密钥和私有密钥复制到一个文件以便在安装该代理时使用。选择关闭。

以下过程会创建也可以写入到 Parameter Store 的 IAM 用户。如果要在 Parameter Store 中存储代理配置文件以使其他服务器可以使用该文件，您需要使用该 IAM 用户。该 IAM 用户提供写入到 Parameter Store 的权限。该用户还提供从实例中读取信息以及将其写入到 CloudWatch 的权限。写入到 Systems Manager Parameter Store 的权限提供了广泛的访问。该 IAM 用户不应附加到您的所有服务器，仅管理员应该使用该用户。只有在 Parameter Store 中存储代理配置文件时，才应使用该 IAM 用户。

创建将配置文件存储到 Parameter Store 中以及向 CloudWatch 发送信息所需的 IAM 用户

1. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择 Users，然后选择 Add user。
3. 为新用户输入用户名。
4. 对于访问类型，请选择编程访问，然后选择下一步：权限。
5. 对于设置权限，请选择直接附加现有策略。
6. 在策略列表中，选中 CloudWatchAgentAdminPolicy 旁边的复选框。如有必要，请使用搜索框查找该策略。
7. 要使用 Systems Manager 安装或配置 CloudWatch 代理，请选中 AmazonSSMManagedInstanceCore 旁边的复选框。此 AWS 托管策略使实例能够使用 Systems Manager 服务核心功能。（如有必要，请使用搜索框查找策略。如果仅通过命令行启动和配置该代理，则不需要使用该策略。）

8. 选择 下一步：标签。
9. （可选）添加一个或多个标签键值对，以组织、跟踪或控制此角色的访问，然后选择 Next: Review (下一步：审核)。
10. 确认列出了正确的策略，然后选择创建用户。
11. 在新用户的行中，选择显示。将访问密钥和私有密钥复制到一个文件以便在安装该代理时使用。选择关闭。

使用 SSM 下载、配置和运行 CloudWatch 代理

本部分介绍了如何使用 Systems Manager 下载代理，然后如何创建您的代理配置文件。您必须确保为 Systems Manager 正确配置实例，然后才能使用 Systems Manager 下载代理。

安装或更新 SSM 代理

在 Amazon EC2 实例上，CloudWatch 代理要求实例运行 SSM 代理 2.2.93.0 或更高版本。在安装 CloudWatch 代理之前，如果还没有在实例上更新或安装 SSM Agent，请执行该操作。

有关在运行 Linux 的实例上安装或更新 SSM Agent 的信息，请参阅 AWS Systems Manager 用户指南中的在 Linux 实例上安装和配置 SSM Agent。

有关安装或更新 SSM Agent 的更多信息，请参阅 AWS Systems Manager 用户指南中的使用 SSM Agent。

验证 Systems Manager 的先决条件

在使用 Systems Manager Run Command 安装和配置 CloudWatch 代理之前，请确保您的实例满足 Systems Manager 的最低要求。有关更多信息，请参阅 AWS Systems Manager 用户指南中的 Systems Manager 的先决条件。

验证 Internet 访问权限

您的 Amazon EC2 实例必须能够连接到 CloudWatch 端点。可以通过互联网网关、NAT 网关或 CloudWatch 接口 VPC 终端节点实现此连接。有关如何配置互联网访问权限的更多信息，请参阅 Amazon VPC 用户指南中的互联网网关。

要在您的代理上配置的终端节点和端口如下所示：

- 如果要使用代理收集指标，则必须允许列出相应区域的 CloudWatch 端点。这些端点在 Amazon Web Services 一般参考 的 Amazon CloudWatch 中列出。

- 如果要使用代理收集日志，则必须允许列出相应区域的 CloudWatch Logs 端点。这些端点在 Amazon Web Services 一般参考 的 Amazon CloudWatch Logs 中列出。
- 如果使用 Systems Manager 安装代理或使用 Parameter Store 存储配置文件，您必须允许列出相应区域的 Systems Manager 端点。这些端点在 Amazon Web Services 一般参考 的 AWS Systems Manager 中列出。

将 CloudWatch 代理程序包下载到您的第一个实例

使用以下步骤来使用 Systems Manager 下载 CloudWatch 代理软件包。

使用 Systems Manager 下载 CloudWatch 代理

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在 Command document（命令文档）列表中，选择 AWS-ConfigureAWSPackage。
5. 在 Targets（目标）区域中，选择要在其上安装 CloudWatch 代理的实例。如果您没有看到特定实例，则可能未将其配置为可使用 Systems Manager 的托管式实例。有关更多信息，请参阅 AWS Systems Manager 用户指南中的为混合环境设置 AWS Systems Manager。
6. 在操作列表中，选择安装。
7. 在 Name (名称) 字段中，输入 *AmazonCloudWatchAgent*。
8. 将 Version (版本) 设置为 latest (最新) 以安装最新版本的代理。
9. 选择运行。
10. （可选）在 Targets and outputs（目标和输出）区域中，选择实例名称旁边的按钮，然后选择 View output（查看输出）。Systems Manager 将显示已成功安装该代理。

创建和修改代理配置文件

在下载 CloudWatch 代理后，您必须先创建配置文件，然后再在任何服务器上启动该代理。

如果您打算在 Systems Manager Parameter Store 中保存代理配置文件，则必须使用 EC2 实例以保存到 Parameter Store。此外，您必须首先使用 `CloudWatchAgentAdminRole` IAM 角色附加到该实例。有关附加角色的更多信息，请参阅《Amazon EC2 用户指南》中的将 IAM 角色附加到实例。

有关创建 CloudWatch 代理配置文件的更多信息，请参阅 创建 CloudWatch 代理配置文件。

使用代理配置在其他 EC2 实例上安装和启动 CloudWatch 代理

在 Parameter Store 中保存 CloudWatch 代理配置后，您可以在其他服务器上安装该代理时使用该配置。

对于每台服务器，请按照本节前面列出的步骤验证 Systems Manager 的先决条件、SSM 代理的版本和互联网访问权限。然后，使用您创建的 CloudWatch 代理配置文件，按照以下说明在其他实例上安装 CloudWatch 代理。

步骤 1：下载和安装 CloudWatch 代理

您需要在将要运行代理的每个服务器上安装代理。CloudWatch 代理在 Amazon Linux 2023 和 Amazon Linux 2 中，可作为软件包使用。如果您使用此操作系统，则可以通过执行以下步骤来使用 Systems Manager 安装该软件包。

> **Note**
>
> 您还必须确保附加到实例的 IAM 角色附加了 CloudWatchAgentServerPolicy。有关更多信息，请参阅 在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。

要使用 Systems Manager 安装 CloudWatch 代理程序包

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在命令文档列表中，选择 AWS-RunShellScript。然后将以下内容粘贴到命令参数中。

```
sudo yum install amazon-cloudwatch-agent
```

5. 选择运行。

在所有支持的操作系统中，您可以使用 Systems Manager Run Command 或 Amazon S3 下载链接下载 CloudWatch 代理程序包。有关使用 Amazon S3 下载链接的信息，请参阅 下载 CloudWatch 代理软件包。

> **Note**
>
> 当您安装或更新 CloudWatch 代理时，仅支持 Uninstall and reinstall（卸载并重新安装）选项。您无法使用 In-place update（就地更新）选项。

Systems Manager Run Command 使您能够管理实例配置。您可以指定一个 Systems Manager 文档，指定一些参数，然后在一个或多个实例上执行命令。实例上的 SSM Agent 负责处理命令并按指定方式配置实例。

使用 Run Command 下载 CloudWatch 代理

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在 Command document（命令文档）列表中，选择 AWS-ConfigureAWSPackage。
5. 在 Targets（目标）区域中，选择要在其上安装 CloudWatch 代理的实例。如果未看到特定的实例，则可能没有为 Run Command 配置该实例。有关更多信息，请参阅 AWS Systems Manager 用户指南中的为混合环境设置 AWS Systems Manager。
6. 在操作列表中，选择安装。
7. 在 Name (名称) 框中，输入 *AmazonCloudWatchAgent*。
8. 将 Version (版本) 设置为 latest (最新) 以安装最新版本的代理。
9. 选择运行。
10. （可选）在 Targets and outputs（目标和输出）区域中，选择实例名称旁边的按钮，然后选择 View output（查看输出）。Systems Manager 将显示已成功安装该代理。

步骤 2：使用您的代理配置文件启动 CloudWatch 代理

使用 Systems Manager Run Command 执行以下步骤来启动代理。

使用 Run Command 启动 CloudWatch 代理

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在命令文档列表中，选择 AmazonCloudWatch-ManageAgent。
5. 在 Targets（目标）区域中，选择安装了 CloudWatch 代理的实例。
6. 在操作列表中，选择配置。
7. 在可选的配置源列表中，选择 ssm。
8. 在可选的配置位置框中，输入您创建并保存到 Systems Manager Parameter Store 的代理配置文件的 Systems Manager 参数名称，如 创建 CloudWatch 代理配置文件 中所述。
9. 在完成这些步骤后，在可选的重新启动列表中选择是以启动该代理。
10. 选择运行。
11. （可选）在 Targets and outputs（目标和输出）区域中，选择实例名称旁边的按钮，然后选择 View output（查看输出）。Systems Manager 将显示已成功启动该代理。

（可选）修改 CloudWatch 代理的通用配置和命名配置文件

CloudWatch 代理包含一个名为 `common-config.toml` 的配置文件。您可以使用该文件指定代理和区域信息（可选）。

在运行 Linux 的服务器上，该文件位于 `/opt/aws/amazon-cloudwatch-agent/etc` 目录中。在运行 Windows Server 的服务器上，该文件位于 `C:\ProgramData\Amazon\AmazonCloudWatchAgent` 目录中。

默认 `common-config.toml` 如下所示：

```
# This common-config is used to configure items used for both ssm and cloudwatch access
```

```
## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

最初将注释所有行。要设置凭证配置文件或代理设置，请从该行中删除 # 并指定一个值。您可以手动编辑该文件，或者使用 Systems Manager 中的 `RunShellScript` Run Command 执行该操作：

- `shared_credential_profile` – 对于本地部署服务器，此行指定 IAM 用户凭证配置文件，以用于将数据发送到 CloudWatch。如果您将此行注释掉，则会使用 `AmazonCloudWatchAgent`。

  在 EC2 实例上，您可以使用此行让 CloudWatch 代理将数据从该实例发送到不同 AWS 区域中的 CloudWatch。要执行此操作，请指定一个包含 `region` 字段的命名配置文件，该字段指定要发送到的区域的名称。

  如果指定 `shared_credential_profile`，您还必须从 `[credentials]` 行开头删除 #。

- `shared_credential_file` – 要让代理在位于默认路径以外的路径中的文件中查找凭证，请在此处指定完整的路径和文件名。在 Linux 上，默认路径为 `/root/.aws`；在 Windows Server 上，默认路径为 `C:\\Users\\Administrator\\.aws`。

  下面的第一个示例显示对 Linux 服务器有效的 `shared_credential_file` 行的语法，第二个示例对 Windows Server 有效。在 Windows Server 上，您必须转义 \ 字符。

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

如果指定 shared_credential_file，您还必须从 [credentials] 行开头删除 #。

- 代理设置 – 如果您的服务器使用 HTTP 或 HTTPS 代理联系 AWS 服务，请在 http_proxy 和 https_proxy 字段中指定这些代理。如果应从代理中排除某些 URL，请在 no_proxy 字段中指定这些 URL 并以逗号分隔。

在本地服务器上安装 CloudWatch 代理

如果已在一台电脑上下载了 CloudWatch 代理并创建了所需的代理配置文件，则可以使用该配置文件在其他本地部署服务器上安装该代理。

在本地部署服务器上下载 CloudWatch 代理

您可以使用 Systems Manager Run Command 或 Amazon S3 下载链接下载 CloudWatch 代理程序包。有关使用 Amazon S3 下载链接的信息，请参阅 下载 CloudWatch 代理软件包。

使用 Systems Manager 下载

要使用 Systems Manager Run Command，您必须通过 Amazon EC2 Systems Manager 注册您的本地部署服务器。有关更多信息，请参阅 AWS Systems Manager 用户指南中的在混合环境中设置 Systems Manager。

如果已注册您的服务器，请将 SSM Agent 更新为最新版本。

有关在运行 Linux 的服务器上更新 SSM Agent 的信息，请参阅 AWS Systems Manager 用户指南中的为混合环境安装 SSM Agent (Linux)。

有关在运行 Windows Server 的服务器上更新 SSM Agent 的信息，请参阅 AWS Systems Manager 用户指南中的为混合环境安装 SSM Agent (Windows)。

使用 SSM Agent 在本地部署服务器上下载 CloudWatch 代理软件包

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。

3. 选择 Run command（运行命令）。
4. 在 Command document（命令文档）列表中，选择 AWS-ConfigureAWSPackage 旁边的按钮。
5. 在 Targets（目标）区域中，选择要在其上安装 CloudWatch 代理的服务器。如果未看到特定的服务器，则可能没有为 Run Command 配置该服务器。有关更多信息，请参阅 AWS Systems Manager 用户指南中的为混合环境设置 AWS Systems Manager。
6. 在操作列表中，选择安装。
7. 在 Name (名称) 框中，输入 *AmazonCloudWatchAgent*。
8. 将 Version (版本) 保留空白以安装最新版本的代理。
9. 选择运行。

   将下载代理软件包，后续步骤是配置并启动该代理。

（安装在本地部署服务器上）指定 IAM 凭证和 AWS 区域

要使 CloudWatch 代理能够从本地部署服务器发送数据，您必须指定先前创建的 IAM 用户的访问密钥和私有密钥。有关创建此用户的更多信息，请参阅创建 IAM 角色和用户以用于 CloudWatch 代理。

您还必须使用 `region` 字段指定要将指标发送到的 AWS 区域。

以下为该文件的示例。

```
[AmazonCloudWatchAgent]
aws_access_key_id=my_access_key
aws_secret_access_key=my_secret_key
region = us-west-1
```

对于 *my_access_key* 和 *my_secret_key*，请使用来自没有写入到 Systems Manager Parameter Store 权限的 IAM 用户的密钥。有关 CloudWatch 代理所需的 IAM 用户的更多信息，请参阅 创建 IAM 用户以用于本地部署服务器上的 CloudWatch 代理。

如果您将此配置文件命名为 `AmazonCloudWatchAgent`，则无需执行任何操作。（可选）您可以为其指定一个不同的名称，并将该名称指定为 `common-config.toml` 文件中的 `shared_credential_profile` 值（将在下一节中介绍）。

下面是使用 aws configure 命令为 CloudWatch 代理创建命名配置文件的示例。该示例假设您使用默认配置文件名称 `AmazonCloudWatchAgent`。

为 CloudWatch 代理创建 AmazonCloudWatchAgent 配置文件

1. 如果您还没有这样做，请在服务器上安装 AWS Command Line Interface。有关更多信息，请参阅安装 AWS CLI。
2. 在 Linux 服务器上，输入以下命令并按照提示进行操作：

```
sudo aws configure --profile AmazonCloudWatchAgent
```

在 Windows Server 上，以管理员身份打开 PowerShell，输入以下命令并按照提示进行操作。

```
aws configure --profile AmazonCloudWatchAgent
```

（可选）修改 CloudWatch 代理的通用配置和命名配置文件

CloudWatch 代理包含一个名为 common-config.toml 的配置文件。您可以（可选）使用该文件指定代理和区域信息。

在运行 Linux 的服务器上，该文件位于 /opt/aws/amazon-cloudwatch-agent/etc 目录中。在运行 Windows Server 的服务器上，该文件位于 C:\ProgramData\Amazon\AmazonCloudWatchAgent 目录中。

默认 common-config.toml 如下所示：

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
```

```
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

最初将注释所有行。要设置凭证配置文件或代理设置，请从该行中删除 # 并指定一个值。您可以手动编辑该文件，或者使用 Systems Manager 中的 RunShellScript Run Command 执行该操作：

- shared_credential_profile – 对于本地部署服务器，此行指定 IAM 用户凭证配置文件，以用于将数据发送到 CloudWatch。如果您将此行注释掉，则会使用 AmazonCloudWatchAgent。有关创建此配置文件的更多信息，请参阅（安装在本地部署服务器上）指定 IAM 凭证和 AWS 区域。

  在 EC2 实例上，您可以使用此行让 CloudWatch 代理将数据从该实例发送到不同 AWS 区域中的 CloudWatch。要执行此操作，请指定一个包含 region 字段的命名配置文件，该字段指定要发送到的区域的名称。

  如果指定 shared_credential_profile，您还必须从 [credentials] 行开头删除 #。

- shared_credential_file – 要让代理在位于默认路径以外的路径中的文件中查找凭证，请在此处指定完整的路径和文件名。在 Linux 上，默认路径为 /root/.aws；在 Windows Server 上，默认路径为 C:\\Users\\Administrator\\.aws。

  下面的第一个示例显示对 Linux 服务器有效的 shared_credential_file 行的语法，第二个示例对 Windows Server 有效。在 Windows Server 上，您必须转义 \ 字符。

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  如果指定 shared_credential_file，您还必须从 [credentials] 行开头删除 #。

- 代理设置 – 如果您的服务器使用 HTTP 或 HTTPS 代理联系 AWS 服务，请在 http_proxy 和 https_proxy 字段中指定这些代理。如果应从代理中排除某些 URL，请在 no_proxy 字段中指定这些 URL 并以逗号分隔。

启动 CloudWatch 代理

您可以使用 Systems Manager Run Command 或命令行启动 CloudWatch 代理。

使用 SSM Agent 在本地部署服务器上启动 CloudWatch 代理

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在命令文档列表中，选择 AmazonCloudWatch-ManageAgent 旁边的按钮。
5. 在目标区域，选择安装了该代理的实例。
6. 在操作列表中，选择配置。
7. 在模式列表中，选择 onPremise。
8. 在 Optional Configuration Location（可选的配置位置）框中，输入您使用向导创建并存储在 Parameter Store 中的代理配置文件的名称。
9. 选择运行。

   该代理使用在配置文件中指定的配置启动。

使用命令行在本地部署服务器上启动 CloudWatch 代理

- 此命令中，`-a fetch-config` 会使代理加载最新版本的 CloudWatch 代理配置文件，`-s` 则会启动该代理。

  Linux：如果在 Systems Manager Parameter Store 中保存了配置文件，请输入以下命令：

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
  ```

  Linux：如果在本地计算机上保存了配置文件，请输入以下命令。将 *configuration-file-path* 替换为指向代理配置文件的路径。如果您使用向导创建此文件，则此文件命名为 `config.json`；如果您手动创建该文件，则该文件可能命名为 `amazon-cloudwatch-agent.json`。

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
  ```

Windows Server：如果在 Systems Manager Parameter Store 中保存代理配置文件，请从 PowerShell 控制台中输入以下命令：

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Windows Server：如果在本地计算机上保存了代理配置文件，请从 PowerShell 控制台中输入以下命令。将 *configuration-file-path* 替换为指向代理配置文件的路径。如果您使用向导创建此文件，则此文件命名为 `config.json`；如果您手动创建该文件，则该文件可能命名为 `amazon-cloudwatch-agent.json`。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c file:configuration-file-path
```

使用 AWS CloudFormation 在新实例上安装 CloudWatch 代理

亚马逊已将若干 AWS CloudFormation 模板上传到 GitHub，以帮助您在新的 Amazon EC2 实例上安装和更新 CloudWatch 代理。有关使用 AWS CloudFormation 的更多信息，请参阅什么是 AWS CloudFormation？。

模板位置为 Deploy the Amazon CloudWatch agent to EC2 instances using AWS CloudFormation（使用 Amazon CloudFormation 将 Amazon CloudWatch 代理部署到 EC2 实例）。此位置既包括 `inline` 又包括 `ssm` 目录。其中每个目录包含同时适用于 Linux 和 Windows 实例的模板。

- `inline` 目录中的模板已将 CloudWatch 代理配置嵌入到 AWS CloudFormation 模板中。默认情况下，Linux 模板收集指标 `mem_used_percent` 和 `swap_used_percent`，Windows 模板收集 `Memory % Committed Bytes In Use` 和 `Paging File % Usage`。

  要修改这些模板以收集不同的指标，请修改模板的以下部分。下面的示例来自 Linux 服务器的模板。按照代理配置文件的格式和语法来进行这些更改。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

```
{
   "metrics":{
      "append_dimensions":{
         "AutoScalingGroupName":"${!aws:AutoScalingGroupName}",
         "ImageId":"${!aws:ImageId}",
```

```
            "InstanceId":"${!aws:InstanceId}",
            "InstanceType":"${!aws:InstanceType}"
        },
        "metrics_collected":{
            "mem":{
                "measurement":[
                    "mem_used_percent"
                ]
            },
            "swap":{
                "measurement":[
                    "swap_used_percent"
                ]
            }
        }
    }
}
```

> **Note**
>
> 在内联模板中，所有占位符变量必须具有感叹号 (!)，然后才能将它们作为一个转义字符。您可以在示例模板中看到此内容。如果您添加其他占位符变量，请务必在名称前添加一个感叹号。

- `ssm` 目录中的模板从 Parameter Store 加载代理配置文件。要使用这些模板，您必须先创建配置文件并将其上传到 Parameter Store。然后，您可以提供模板中的文件的 Parameter Store 名称。您可以手动或使用向导来创建配置文件。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。

您可以使用这两种类型的模板安装 CloudWatch 代理和更新代理配置。

教程：使用 AWS CloudFormation 内联模板安装和配置 CloudWatch 代理

本教程将演练如何使用 AWS CloudFormation 在新的 Amazon EC2 实例上安装 CloudWatch 代理。本教程使用内联模板在运行 Amazon Linux 2 的新实例上进行安装，这不要求使用 JSON 配置文件或 Parameter Store。内联模板在模板中包含代理配置。在本教程中，您将使用模板中包含的默认代理配置。

在介绍安装代理的过程之后，该教程继续说明如何更新代理。

使用 AWS CloudFormation 在新实例上安装 CloudWatch 代理

1. 从 GitHub 下载模板。在本教程中，下载 Amazon Linux 2 的内联模板，如下所示：

```
curl -O https://raw.githubusercontent.com/aws-cloudformation/aws-cloudformation-templates/main/Solutions/AmazonCloudWatchAgent/inline/amazon_linux.yaml
```

2. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
3. 选择创建堆栈。
4. 对于选择一个模板，选择将模板上传到 Amazon S3，选择下载的模板，然后选择下一步。
5. 在 Specify Details (指定详细信息) 页面上，填写以下参数，然后选择 Next (下一步)：
   - 堆栈名称：为 AWS CloudFormation 堆栈选择堆栈名称。
   - IAMRole：选择一个 IAM 角色，该角色有权写入 CloudWatch 指标、日志和跟踪信息。有关更多信息，请参阅 在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。
   - InstanceAMI：选择在您将要启动堆栈的区域中有效的 AMI。
   - InstanceType：选择有效的实例类型。
   - KeyName：要对新实例启用 SSH 访问，请选择一个现有 Amazon EC2 密钥对。如果还没有 Amazon EC2 密钥对，可以在 AWS Management Console 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。
   - SSHLocation：指定可用于通过 SSH 连接到实例的 IP 地址范围。默认情况下，允许从任何 IP 地址进行访问。
6. 在 Options (选项) 页面上，您可以选择为堆栈资源添加标签。选择下一步。
7. 在 Review（审核）页面上，审核您的信息，确认堆栈可创建 IAM 资源，然后选择 Create（创建）。

   如果您刷新控制台，您将看到新堆栈具有 CREATE_IN_PROGRESS 状态。
8. 创建实例时，您可以在 Amazon EC2 控制台中看到它。（可选）您可以连接到主机并检查进度。

   使用以下命令确认代理已安装：

```
rpm -qa amazon-cloudwatch-agent
```

   使用以下命令确认代理正在运行：

```
ps aux | grep amazon-cloudwatch-agent
```

以下过程演示如何使用 AWS CloudFormation 通过内联模板更新 CloudWatch 代理。默认内联模板收集 mem_used_percent 指标。在本教程中，您将更改代理配置以停止收集该指标。

使用 AWS CloudFormation 更新 CloudWatch 代理

1. 在上一个过程中下载的模板中，删除以下行，然后保存该模板：

```
"mem": {

     "measurement": [
         "mem_used_percent"
       ]
 },
```

2. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
3. 在 AWS CloudFormation 控制面板中，选择您创建的堆栈，然后选择 Update Stack (更新堆栈)。
4. 对于选择模板，选择将模板上传到 Amazon S3，选择修改的模板，然后选择下一步。
5. 在 Options (选项) 页面上，选择 Next (下一步)，然后选择 Next (下一步)。
6. 在审核页面上，审核您的信息，然后选择更新。

   经过一段时间后，您会看到 UPDATE_COMPLETE。

教程：使用 AWS CloudFormation 和 Parameter Store 安装 CloudWatch

本教程将演练如何使用 AWS CloudFormation 在新的 Amazon EC2 实例上安装 CloudWatch 代理。本教程使用您在 Parameter Store 中创建并保存的代理配置文件，在运行 Amazon Linux 2 的新实例上进行安装。

在介绍安装代理的过程之后，该教程继续说明如何更新代理。

使用 Parameter Store 中的配置通过 AWS CloudFormation 在新实例上安装 CloudWatch 代理

1. 如果您尚未执行此操作，请将 CloudWatch 代理软件包下载到您的某台电脑上，以便您可以创建代理配置文件。有关使用 Parameter Store 下载代理的更多信息，请参阅 使用 SSM 下载、配置和运行 CloudWatch 代理。有关使用命令行下载软件包的更多信息，请参阅使用命令行下载和配置 CloudWatch 代理。
2. 创建代理配置文件并将其保存在 Parameter Store 中。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。

3. 从 GitHub 下载模板，如下所示：

```
curl -O https://raw.githubusercontent.com/awslabs/aws-cloudformation-templates/
master/aws/solutions/AmazonCloudWatchAgent/ssm/amazon_linux.template
```

4. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
5. 选择创建堆栈。
6. 对于选择一个模板，选择将模板上传到 Amazon S3，选择您下载的模板，然后选择下一步。
7. 在 Specify Details (指定详细信息) 页面上，相应填写以下参数，然后选择 Next (下一步)：
   - 堆栈名称：为 AWS CloudFormation 堆栈选择堆栈名称。
   - IAMRole：选择一个 IAM 角色，该角色有权写入 CloudWatch 指标、日志和跟踪信息。有关更多信息，请参阅 在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。
   - InstanceAMI：选择在您将要启动堆栈的区域中有效的 AMI。
   - InstanceType：选择有效的实例类型。
   - KeyName：要对新实例启用 SSH 访问，请选择一个现有 Amazon EC2 密钥对。如果还没有 Amazon EC2 密钥对，可以在 AWS Management Console 中创建一个。有关更多信息，请参阅《Amazon EC2 用户指南》中的 Amazon EC2 密钥对。
   - SSHLocation：指定可用于通过 SSH 连接到实例的 IP 地址范围。默认情况下，允许从任何 IP 地址进行访问。
   - SSMKey：指定在 Parameter Store 中创建并保存的代理配置文件。
8. 在 Options (选项) 页面上，您可以选择为堆栈资源添加标签。选择下一步。
9. 在 Review（审核）页面上，审核您的信息，确认堆栈可创建 IAM 资源，然后选择 Create（创建）。

   如果您刷新控制台，您将看到新堆栈具有 CREATE_IN_PROGRESS 状态。
10. 创建实例时，您可以在 Amazon EC2 控制台中看到它。（可选）您可以连接到主机并检查进度。

    使用以下命令确认代理已安装：

```
rpm -qa amazon-cloudwatch-agent
```

    使用以下命令确认代理正在运行：

```
ps aux | grep amazon-cloudwatch-agent
```

以下过程演示如何使用 AWS CloudFormation，通过您在 Parameter Store 中保存的代理配置更新 CloudWatch 代理。

通过 AWS CloudFormation 使用 Parameter Store 中的配置更新 CloudWatch 代理

1. 将存储在 Parameter Store 中的代理配置文件更改为所需的新配置。
2. 在您于the section called “教程：使用 AWS CloudFormation 和 Parameter Store 安装 CloudWatch”主题中下载的 AWS CloudFormation 模板中更改版本号。例如，您可能将 `VERSION=1.0` 更改为 `VERSION=2.0`。
3. 打开 AWS CloudFormation 控制台，地址：https://console.aws.amazon.com/cloudformation。
4. 在 AWS CloudFormation 控制面板中，选择您创建的堆栈，然后选择 Update Stack (更新堆栈)。
5. 对于选择模板，选择将模板上传到 Amazon S3，选择您刚刚修改的模板，然后选择下一步。
6. 在 Options (选项) 页面上，选择 Next (下一步)，然后选择 Next (下一步)。
7. 在审核页面上，审核您的信息，然后选择更新。

   经过一段时间后，您会看到 `UPDATE_COMPLETE`。

对利用 AWS CloudFormation 的 CloudWatch 代理安装进行故障排除

本节帮助您排查使用 AWS CloudFormation 安装和更新 CloudWatch 代理时出现的问题。

检测何时更新失败

如果您使用 AWS CloudFormation 更新您的 CloudWatch 代理配置并使用无效的配置，代理将停止向 CloudWatch 发送任何指标。检查代理配置更新是否成功的快速方法是查看 `cfn-init-cmd.log` 文件。在 Linux 服务器上，此文件位于 `/var/log/cfn-init-cmd.log`。在 Windows 实例上，此文件位于 `C:\cfn\log\cfn-init-cmd.log`。

指标缺失

如果您在安装或更新代理后没有看到预期会看到的指标，请确认代理配置为收集该指标。要执行此操作，请检查 `amazon-cloudwatch-agent.json` 文件，以确保该指标已列出，并且检查您是否可以在正确的指标命名空间中看到它。有关更多信息，请参阅 CloudWatch 代理文件和位置。

CloudWatch 代理凭证首选项

本节概述了 CloudWatch 代理在与其他 AWS 服务和 API 通信时用来获取凭证的凭证提供程序链。顺序如下所示。以下列表中第二到第五项列出的首选项与 AWS SDK 中定义的首选项顺序相同。有关更多信息，请参阅 SDK 文档中的指定凭证。

1. 共享配置和凭证文件如 CloudWatch 代理的 `common-config.toml` 文件中所定义。有关更多信息，请参阅（可选）修改代理的通用配置或区域信息。
2. AWS SDK 环境变量

> **Important**
>
> 在 Linux 上，如果您使用 `amazon-cloudwatch-agent-ctl` 脚本运行 CloudWatch 代理，则该脚本会将代理作为 `systemd` 服务启动。在这种情况下，代理无法访问诸如 `HOME`、`AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 之类的环境变量。

3. 共享配置和凭证文件位于 `$HOME/%USERPROFILE%` 中

> **Note**
>
> 对于 Linux 和 MacOS，CloudWatch 代理会在 `$HOME` 中查找 `.aws/credentials`，而对于 Windows，则会在 `%USERPROFILE%` 中查找。与 AWS SDK 不同的是，如果无法访问环境变量，则 CloudWatch 代理没有用于确定主目录的备用方法。这种行为上的差异是为了保持与 AWS SDK 早期实现的向后兼容性。
>
> 此外，与 `common-config.toml` 中提供的共享凭证不同的是，如果 AWS SDK 派生的共享凭证过期并进行轮换，则 CloudWatch 代理不会自动获取续订的凭证，需要重启代理才能获取该凭证。

4. 任务的 AWS Identity and Access Management 角色，如果存在一个应用程序使用 Amazon Elastic Container Service 任务定义或 RunTask API 操作。
5. 附加到 Amazon EC2 实例的实例配置文件

作为最佳实践，建议您在使用 CloudWatch 代理时按以下顺序指定凭证。

1. 如果您的应用程序使用 Amazon Elastic Container Service 任务定义或 RunTask API 操作，则请使用 IAM 角色执行任务。
2. 如果您的应用程序在 Amazon EC2 实例上运行，请使用 IAM 角色。

3. 使用 CloudWatch 代理 `common-config.toml` 文件指定凭证文件。此凭证文件与其他 AWS SDK 和 AWS CLI 使用的凭证文件相同。如果您已经在使用共享凭证文件，则也可以使用该文件来实现此目的。如果您使用 CloudWatch 代理的 `common-config.toml` 文件提供该凭证，则可以确保代理在这些凭证过期时使用轮换后的凭证，进行替换而无需重启代理。
4. 使用环境变量。如果您是在 Amazon EC2 实例以外的计算机上进行开发工作，则设置环境变量非常有用。

> **Note**
>
> 如果您按照 向不同账户发送指标、日志和跟踪信息 中的说明将遥测数据发送到不同的账户，则 CloudWatch 代理将使用本节所述的凭证提供程序链来获取初始凭证集。然后，它在担任 CloudWatch 代理配置文件中的 `role_arn` 指定的 IAM 角色时使用这些凭证。

验证 CloudWatch 代理软件包的签名

包含 GPG 签名文件以用于 Linux 服务器上的 CloudWatch 代理软件包。您可以使用公有密钥验证该代理下载文件是否为原始的而未进行修改。

对于 Windows Server，您可以使用 MSI 验证签名。

对于 macOS 电脑，代理下载软件包中包含签名。

要查找正确的签名文件，请参阅下表。对于每个架构和操作系统，有一个常规链接以及对应于每个区域的链接。例如，对于 Amazon Linux 2023 和 Amazon Linux 2 以及 x86-64 架构，三个有效的链接如下：

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

> **Note**
>
> 要下载 CloudWatch 代理，您的连接必须使用 TLS 1.2 或更高版本。

架构	平台	下载链接	签名文件链接
x86-64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazon	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/amd64/latest/a

架构	平台	下载链接	签名文件链接
		s.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	mazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazons.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazons.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazons.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig

架构	平台	下载链接	签名文件链接
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig

架构	平台	下载链接	签名文件链接
ARM64	Amazon Linux 2023 和 Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig

架构	平台	下载链接	签名文件链接
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

验证 Linux 服务器上的 CloudWatch 代理软件包

1. 下载公有密钥。

```
shell$ wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-
cloudwatch-agent.gpg
```

2. 将公有密钥导入到您的密钥环中。

```
shell$  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

请记下密钥值，因为需要在下一步中使用该值。在上一示例中，键值为 3B789C72。

3. 通过运行以下命令，将 *key-value* 替换为上一步中的值来验证指纹：

```
shell$  gpg --fingerprint key-value
pub   2048R/3B789C72 2017-11-14
      Key fingerprint = 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid                  Amazon CloudWatch Agent
```

指纹字符串应等于以下内容：

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

如果指纹字符串不匹配，请不要安装该代理。请联系 Amazon Web Services。

在验证指纹后，您可以使用该指纹验证 CloudWatch 代理软件包的签名。

4. 使用 wget 下载软件包签名文件。要确定正确的签名文件，请参阅前一个表。

```
wget Signature File Link
```

5. 要验证签名，请运行 gpg --verify。

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

如果输出包含短语 BAD `signature`，则检查是否正确执行了此过程。如果您继续获得该响应，请与 Amazon Web Services 联系，并避免使用已下载的文件。

请注意有关信任的警告。只有当您或您信任的某个人对密钥进行了签名，密钥才是可信的。这并不意味着签名无效，只是您尚未验证公有密钥而已。

验证运行 Windows Server 的服务器上的 CloudWatch 代理软件包

1. 从 https://gnupg.org/download/ 中下载并安装适用于 Windows 的 GnuPG。在安装时，请包含 Shell 扩展 (GpgEx) 选项。

   您可以在 Windows PowerShell 中执行其余步骤。

2. 下载公有密钥。

```
PS> wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-cloudwatch-
agent.gpg -OutFile amazon-cloudwatch-agent.gpg
```

3. 将公有密钥导入到您的密钥环中。

```
PS>  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

请记下密钥值，因为需要在下一步中使用该值。在上一示例中，键值为 3B789C72。

4. 通过运行以下命令，将 *key-value* 替换为上一步中的值来验证指纹：

```
PS>  gpg --fingerprint key-value
pub   rsa2048 2017-11-14 [SC]
      9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid           [ unknown] Amazon CloudWatch Agent
```

指纹字符串应等于以下内容：

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

如果指纹字符串不匹配，请不要安装该代理。请联系 Amazon Web Services。

在验证指纹后，您可以使用该指纹验证 CloudWatch 代理软件包的签名。

5. 使用 wget 下载软件包签名文件。要确定正确的签名文件，请参阅 CloudWatch 代理下载链接。
6. 要验证签名，请运行 gpg --verify。

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

如果输出包含短语 BAD signature，则检查是否正确执行了此过程。如果您继续获得该响应，请与 Amazon Web Services 联系，并避免使用已下载的文件。

请注意有关信任的警告。只有当您或您信任的某个人对密钥进行了签名，密钥才是可信的。这并不意味着签名无效，只是您尚未验证公有密钥而已。

验证 macOS 电脑上的 CloudWatch 代理软件包

- 在 macOS 上进行签名验证的方法有两种。
  - 通过运行以下命令验证指纹。

```
pkgutil --check-signature amazon-cloudwatch-agent.pkg
```

您应该会看到类似以下内容的结果。

```
Package "amazon-cloudwatch-agent.pkg":
        Status: signed by a developer certificate issued by Apple for
 distribution
        Signed with a trusted timestamp on: 2020-10-02 18:13:24 +0000
        Certificate Chain:
        1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
        Expires: 2024-10-18 22:31:30 +0000
        SHA256 Fingerprint:
        81 B4 6F AF 1C CA E1 E8 3C 6F FB 9E 52 5E 84 02 6E 7F 17 21 8E FB
        0C 40 79 13 66 8D 9F 1F 10 1C

 ------------------------------------------------------------------------
        2. Developer ID Certification Authority
        Expires: 2027-02-01 22:12:15 +0000
        SHA256 Fingerprint:
        7A FC 9D 01 A6 2F 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 8D 03
        F2 9C 88 CF B0 B1 BA 63 58 7F

 ------------------------------------------------------------------------
        3. Apple Root CA
        Expires: 2035-02-09 21:40:36 +0000
        SHA256 Fingerprint:
        B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C
        68 C5 BE 91 B5 A1 10 01 F0 24
```

- 或者，下载并使用 sig 格式文件。若要使用此方法，请按照下列步骤操作。
  - 通过输入以下命令，将 GPG 应用程序安装到您的 macOS 主机。

    ```
    brew install GnuPG
    ```

- 使用 curl 下载软件包签名文件。要确定正确的签名文件，请参阅 CloudWatch 代理下载链接。
- 要验证签名，请运行 gpg --verify。

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

```
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

如果输出包含短语 `BAD signature`，则检查是否正确执行了此过程。如果您继续获得该响应，请与 Amazon Web Services 联系，并避免使用已下载的文件。

请注意有关信任的警告。只有当您或您信任的某个人对密钥进行了签名，密钥才是可信的。这并不意味着签名无效，只是您尚未验证公有密钥而已。

创建 CloudWatch 代理配置文件

在任何服务器上运行 CloudWatch 代理之前，必须创建一个或多个 CloudWatch 代理配置文件。

代理配置文件是一个 JSON 文件，它指定了该代理要收集的指标、日志和跟踪信息，包括自定义指标。您可以使用向导创建该文件，或者从 Scratch 自行创建该文件。您还可以最初使用向导创建配置文件，然后手动对其进行修改。如果手动创建或修改该文件，该过程更复杂一些，但您可以更好地控制收集的指标，并且可以指定不通过向导提供的指标。

每次更改代理配置文件时，您必须重新启动该代理以使更改生效。要重新启动该代理，请按照（可选）修改 CloudWatch 代理的通用配置和命名配置文件中的说明进行操作。

在创建配置文件后，可以将其手动保存为 JSON 文件，然后在服务器上安装代理时使用该文件。或者，如果您打算在服务器上安装代理时使用 Systems Manager，则可以将该文件存储在 Systems Manager Parameter Store 中。

CloudWatch 代理支持使用多个配置文件。有关更多信息，请参阅 多个 CloudWatch 代理配置文件。

CloudWatch 代理收集的指标、日志和跟踪记录会产生费用。有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

内容

- 使用向导创建 CloudWatch 代理配置文件
- 手动创建或编辑 CloudWatch 代理配置文件

使用向导创建 CloudWatch 代理配置文件

代理配置文件向导 `amazon-cloudwatch-agent-config-wizard` 会询问一系列问题，帮助您根据需要配置 CloudWatch 代理。

必需的凭证

如果在启动向导之前已具有 AWS 凭证和配置文件，向导可以自动检测要使用的凭证和 AWS 区域。有关这些文件的更多信息，请参阅 AWS Systems Manager 用户指南中的配置和凭证文件。

在 AWS 凭证文件中，向导会检查默认凭证，还会查找 `AmazonCloudWatchAgent` 部分，如下所示：

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

向导将显示默认凭证、来自 `AmazonCloudWatchAgent` 的凭证以及 `Others` 选项。您可以选择要使用的凭证。如果您选择 `Others`，则可以输入凭证。

对于 *my_access_key* 和 *my_secret_key*，请使用来自具有写入到 Systems Manager Parameter Store 权限的 IAM 用户的密钥。有关 CloudWatch 代理所需的 IAM 用户的更多信息，请参阅 创建 IAM 用户以用于本地部署服务器上的 CloudWatch 代理。

在 AWS 配置文件中，您可以指定代理将指标发送到的区域（如果与 `[default]` 部分不同）。默认设置为将指标发布到 Amazon EC2 实例所在的区域。如果要将指标发布到不同的区域，请在此处指定该区域。在以下示例中，指标将发布到 `us-west-1` 区域。

```
[AmazonCloudWatchAgent]
region = us-west-1
```

运行 CloudWatch 代理配置向导

创建 CloudWatch 代理配置文件

1. 输入以下命令以启动 CloudWatch 代理配置向导：

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
   ```

   在运行 Windows Server 的服务器上，运行以下命令，以启动向导：

   ```
   cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
   ```

   ```
   .\amazon-cloudwatch-agent-config-wizard.exe
   ```

2. 回答这些问题以自定义您的服务器的配置文件。
3. 如果您要在本地存储配置文件，则配置文件 `config.json` 将存储在 Linux 服务器上的 `/opt/aws/amazon-cloudwatch-agent/bin/` 中以及 Windows 服务器上的 `C:\Program Files\Amazon\AmazonCloudWatchAgent` 中。然后，您可以将该文件复制到要安装代理的其他服务器上。

   如果要使用 Systems Manager 安装和配置该代理，请务必在提示是否将该文件存储在 Systems Manager Parameter Store 时回答 Yes（是）。您也可以选择将该文件存储在 Parameter Store 中，即使未使用 SSM Agent 安装 CloudWatch 代理。要能够将该文件存储在 Parameter Store 中，您必须使用具有足够权限的 IAM 角色。有关更多信息，请参阅 创建 IAM 角色和用户以用于 CloudWatch 代理。

CloudWatch 代理预定义指标集

向导配置了具有不同详细信息级别的预定义指标集。下表显示了这些指标集。有关这些指标的更多信息，请参阅 CloudWatch 代理收集的指标。

> **Note**
>
> Parameter Store 支持标准层和高级层中的参数。这些参数层与这些表中描述的“基本”、“标准”和“高级”指标详细信息级别无关。

运行 Linux 的 Amazon EC2 实例

详细信息级别	包含的指标
基本	Mem：mem_used_percent Disk：disk_used_percent `disk` 指标（如 `disk_used_percent`）具有一个用于 `Partition` 的维度，这意味着生成的自定义指标的数量取决于与您的实例关联的分区数量。您拥有的磁盘分区数取决于您使用的 AMI 以及您附加到服务器的 Amazon EBS 卷数。
Standard	CPU：`cpu_usage_idle`、`cpu_usage_iowait`、`cpu_usage_user`、`cpu_usage_system`

详细信息级别	包含的指标
	Disk：disk_used_percent 、disk_inodes_free Diskio：diskio_io_time Mem：mem_used_percent Swap：swap_used_percent
高级	CPU：cpu_usage_idle 、cpu_usage_iowait 、cpu_usage_user 、cpu_usage_system Disk：disk_used_percent 、disk_inodes_free Diskio：diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem：mem_used_percent Netstat：netstat_tcp_established 、netstat_tcp_time_wait Swap：swap_used_percent

运行 Linux 的本地服务器

详细信息级别	包含的指标
基本	Disk：disk_used_percent Diskio：diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem：mem_used_percent Net：net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv Swap：swap_used_percent
Standard	CPU：cpu_usage_idle 、cpu_usage_iowait

详细信息级别	包含的指标
	Disk：disk_used_percent 、disk_inodes_free Diskio：diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem：mem_used_percent Net：net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv Swap：swap_used_percent
高级	CPU： cpu_usage_guest 、cpu_usage_idle 、cpu_usage_iowait 、cpu_usage_steal 、cpu_usage_user 、cpu_usage_system Disk：disk_used_percent 、disk_inodes_free Diskio：diskio_io_time 、diskio_write_bytes 、diskio_read_bytes 、diskio_writes 、diskio_reads Mem：mem_used_percent Net：net_bytes_sent 、net_bytes_recv 、net_packets_sent 、net_packets_recv Netstat：netstat_tcp_established 、netstat_tcp_time_wait Swap：swap_used_percent

运行 Windows Server 的 Amazon EC2 实例

> **Note**
>
> 此表中列出的指标名称显示了在控制台中查看时该指标的显示方式。实际的指标名称可能不包含第一个单词。例如，LogicalDisk % Free Space 的实际指标名称仅为 % Free Space。

详细信息级别	包含的指标
基本	Memory：Memory % Committed Bytes In Use LogicalDisk：LogicalDisk % Free Space
Standard	Memory：Memory % Committed Bytes In Use Paging：Paging File % Usage Processor：Processor % Idle Time、Processor % Interrupt Time、Processor % User Time PhysicalDisk：PhysicalDisk % Disk Time LogicalDisk：LogicalDisk % Free Space
高级	Memory：Memory % Committed Bytes In Use Paging：Paging File % Usage Processor：Processor % Idle Time、Processor % Interrupt Time、Processor % User Time LogicalDisk：LogicalDisk % Free Space PhysicalDisk：PhysicalDisk % Disk Time 、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec TCP：TCPv4 Connections Established 、TCPv6 Connections Established

运行 Windows Server 的本地服务器

> **Note**
>
> 此表中列出的指标名称显示了在控制台中查看时该指标的显示方式。实际的指标名称可能不包含第一个单词。例如，LogicalDisk % Free Space 的实际指标名称仅为 % Free Space。

详细信息级别	包含的指标
基本	Paging：Paging File % Usage Processor：Processor % Processor Time LogicalDisk：LogicalDisk % Free Space PhysicalDisk：、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec Memory：Memory % Committed Bytes In Use Network Interface：Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec
Standard	Paging：Paging File % Usage Processor：Processor % Processor Time、Processor % Idle Time、Processor % Interrupt Time LogicalDisk：LogicalDisk % Free Space PhysicalDisk：PhysicalDisk % Disk Time 、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec Memory：Memory % Committed Bytes In Use

详细信息级别	包含的指标
	Network Interface：Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec
高级	Paging：Paging File % Usage Processor：Processor % Processor Time、Processor % Idle Time、Processor % Interrupt Time、Processor % User Time LogicalDisk：LogicalDisk % Free Space PhysicalDisk：PhysicalDisk % Disk Time 、PhysicalDisk Disk Write Bytes/sec 、PhysicalDisk Disk Read Bytes/sec 、PhysicalDisk Disk Writes/sec 、PhysicalDisk Disk Reads/sec Memory：Memory % Committed Bytes In Use Network Interface：Network Interface Bytes Sent/sec、Network Interface Bytes Received/sec 、Network Interface Packets Sent/sec、Network Interface Packets Received/sec TCP：TCPv4 Connections Established 、TCPv6 Connections Established

手动创建或编辑 CloudWatch 代理配置文件

CloudWatch 代理配置文件是一个 JSON 文件，其中包含四个部分：agent、metrics、logs 和 traces，如下所述：

- agent 部分包含总体代理配置的字段。
- metrics 部分指定要收集并发布到 CloudWatch 的自定义指标。如果只将代理用于收集日志，则可以忽略文件中的 metrics 部分。
- logs 部分指定要将哪些日志文件发布到 CloudWatch Logs。如果服务器运行 Windows Server，这可能包括 Windows 事件日志中的事件。
- traces 部分指定收集并发送到 AWS X-Ray 的跟踪信息的来源。

以下几节介绍了该 JSON 文件的结构和字段。您还可以查看该配置文件的架构定义。在 Linux 服务器上，架构定义位于 *installation-directory*/doc/amazon-cloudwatch-agent-schema.json 中；在运行 Windows Server 的服务器上，架构定义位于 *installation-directory*/amazon-cloudwatch-agent-schema.json 中。

如果手动创建或编辑 代理配置文件，则可以给它指定任何名称。为了简化故障排除，我们建议您在 Linux 服务器上将其命名为 /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json，在运行 Windows Server 的服务器上将其命名为 $Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json。创建该文件后，您可以将其复制到要安装代理的其他服务器上。

> **Note**
>
> CloudWatch 代理收集的指标、日志和跟踪记录会产生费用。有关定价的更多信息，请参阅 Amazon CloudWatch 定价。

CloudWatch 代理配置文件：Agent 部分

agent 部分包含以下字段。向导不会创建 agent 部分。相反，向导忽略该部分，并在该部分的所有字段中使用默认值。

- metrics_collection_interval – 可选。指定该配置文件中指定的所有指标的收集频率。您可以为特定类型的指标覆盖该值。

  该值是以秒为单位指定的。例如，指定 10 将导致每 10 秒收集一次指标；将其设置为 300，将指定每 5 分钟收集一次指标。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。

  默认值是 60。

- region – 指定在监控 Amazon EC2 实例时用于 CloudWatch 端点的区域。收集的指标将发送到该区域，例如，us-west-1。如果省略该字段，则该代理将指标发送到 Amazon EC2 实例所在的区域。

  如果监控本地服务器，则不使用该字段，该代理从 AWS 配置文件的 AmazonCloudWatchAgent 配置文件中读取区域。

- credentials – 指定一个在向不同 AWS 账户发送指标、日志和跟踪信息时使用的 IAM 角色。如果指定，则此字段包含一个参数 role_arn。
  - role_arn – 指定一个在向不同 AWS 账户发送指标、日志和跟踪信息时用于身份验证的 IAM 角色的 Amazon 资源名称（ARN）。有关更多信息，请参阅 向不同账户发送指标、日志和跟踪信息。
- debug – 可选。指定使用调试日志消息运行 CloudWatch 代理。默认值为 false。
- aws_sdk_log_level – 可选。仅在 CloudWatch 代理版本 1.247350.0 及更高版本中受支持。

  您可以将此字段指定为以让代理为 AWS SDK 终端节点执行日志记录。此字段的值可以包括以下一个或多个选项。使用 | 字符分隔多个选项。
  - LogDebug
  - LogDebugWithSigning
  - LogDebugWithHTTPBody
  - LogDebugRequestRetries
  - LogDebugWithEventStreamBody

  有关这些选项的更多信息，请参阅 LogLevelType。
- logfile – 指定 CloudWatch 代理将日志消息写入到的位置。如果指定空字符串，则将日志传输到 stderr。如果未指定该选项，则默认位置如下所示：
  - Linux：/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log
  - Windows Server：c:\\ProgramData\\Amazon\\CloudWatchAgent\\Logs\\amazon-cloudwatch-agent.log

  CloudWatch 代理自动轮换它创建的日志文件。在日志文件大小达到 100 MB 时，将会轮换掉该文件。代理将轮换的日志文件最多保留七天，并最多保留五个轮换掉的备份日志文件。备份日志文件的文件名后附有时间戳。时间戳显示轮换掉文件的日期和时间，例如，amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz。
- omit_hostname – 可选。默认情况下，主机名将作为代理收集的指标维度进行发布（除非您使用 metrics 部分的 append_dimensions 字段）。将 omit_hostname 设置为 true，以禁止将主机名作为维度进行发布（即使在您没有使用 append_dimensions 的情况下）。默认值为 false。
- run_as_user – 可选。指定用于运行 CloudWatch 代理的用户。如果未指定该参数，则使用 root 用户。该选项仅在 Linux 服务器上有效。

如果指定该选项，在启动 CloudWatch 代理之前，用户必须存在。有关更多信息，请参阅 以不同用户身份运行 CloudWatch 代理。

- user_agent – 可选。指定 user-agent 字符串，该字符串由 CloudWatch 代理在 CloudWatch 后端进行 API 调用时使用。默认值是由代理版本、用于编译代理的 Go 编程语言版本、运行时操作系统和架构、构建时间以及启用的插件组成的字符串。
- usage_data – 可选。默认情况下，每当向 CloudWatch 发布指标或日志时，CloudWatch 代理都会向 CloudWatch 发送自身的运行状况和性能数据。这些数据不会产生任何费用。您可以通过将 usage_data 指定为 false，来防止代理发送这些数据。如果省略此参数，则将使用 true 的默认值，并且代理会发送运行状况和性能数据。

  如果将此值设置为 false，则必须停止并重新启动代理才能使其生效。

下面是 agent 部分的示例。

```
"agent": {
   "metrics_collection_interval": 60,
   "region": "us-west-1",
   "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
   "debug": false,
   "run_as_user": "cwagent"
  }
```

CloudWatch 代理配置文件：Metrics（指标）部分

Linux 和 Windows 的通用字段

在运行 Linux 或 Windows Server 的服务器上，metrics 部分包含以下字段：

- namespace – 可选。用于该代理收集的指标的命名空间。默认值为 CWAgent。最大长度为 255 个字符。以下是 示例：

```
{
  "metrics": {
    "namespace": "Development/Product1Metrics",
   ......
   },
}
```

- append_dimensions – 可选。将 Amazon EC2 指标维度添加到该代理收集的所有指标中。这也会导致代理不会将主机名作为维度发布。

  以下列表中显示了受 append_dimensions 支持的键值对。任何其他密钥值对都将被忽略。代理支持与下表所列完全一致的键值对。您无法通过更改键值来发布其他维度名称。

  - "ImageId":"${aws:ImageId}" 将实例的 AMI ID 设置为 ImageId 维度的值。
  - "InstanceId":"${aws:InstanceId}" 将实例的实例 ID 设置为 InstanceId 维度的值。
  - "InstanceType":"${aws:InstanceType}" 将实例的实例类型设置为 InstanceType 维度的值。
  - "AutoScalingGroupName":"${aws:AutoScalingGroupName}" 将实例的 Auto Scaling 组名称设置为 AutoScalingGroupName 维度的值。

  如果您想将维度附加到具有任意键值对的指标，请在该字段中为该特定类型的指标使用 append_dimensions 参数。

  如果您指定一个值（该值取决于 Amazon EC2 元数据），并且使用代理，则必须确保服务器能够访问 Amazon EC2 的端点。有关这些端点的更多信息，请参阅 Amazon Web Services 一般参考 中的 Amazon Elastic Compute Cloud（Amazon EC2）。

- aggregation_dimensions – 可选。指定所收集的指标要在其上聚合的维度。例如，如果累积 AutoScalingGroupName 维度上的指标，则会汇总每个 Auto Scaling 组的所有实例中的指标，并将这些指标作为一个整体进行查看。

  您可以累积一个或多个维度的指标。例如，指定 [["InstanceId"], ["InstanceType"], ["InstanceId","InstanceType"]] 将单独汇总实例 ID 的指标，单独汇总实例类型的指标以及汇总两个维度组合的指标。

  您也可以指定 [] 以将所有指标累积到一个集合中，而不考虑任何维度。

- endpoint_override – 指定一个 FIPS 终端节点或私有链接，以用作代理在其中发送指标的端点。如果指定此项并设置私有链接，您可以将指标发送到 Amazon VPC 终端节点。有关更多信息，请参阅 Amazon VPC 是什么？。

  endpoint_override 的值必须是表示 URL 的字符串。

  例如，在配置文件的指标部分中，以下内容将代理设置为在发送指标时使用 VPC 终端节点。

```
{
  "metrics": {
```

```
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.monitoring.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- `metrics_collected` – 必需。指定要收集的指标，包括通过 StatsD 或 collectd 收集的自定义指标。该部分包含几个小节。

  `metrics_collected` 部分的内容取决于该配置文件适用于运行 Linux 还是 Windows Server 的服务器。

- `force_flush_interval` – 以秒为单位指定指标在发送到服务器之前保留在内存缓冲区中的最大时间量。无论此处的设置如何，如果缓冲区中的指标大小达到 1MB 或 1000 个不同的指标，这些指标会立即发送到服务器。

  默认值是 60。

- `credentials` – 指定一个在向不同账户发送指标时使用的 IAM 角色。如果指定，则此字段包含一个参数 `role_arn`。
  - `role_arn` – 指定一个在向不同账户发送指标时用于身份验证的 IAM 角色的 ARN。有关更多信息，请参阅 向不同账户发送指标、日志和跟踪信息。如果在此指定，该值将覆盖在配置文件的 `agent` 部分指定的 `role_arn`（如果有）。

Linux 部分

在运行 Linux 的服务器上，配置文件的 `metrics_collected` 部分可能还包含以下字段。

其中的很多字段可能包含 `measurement` 部分，其中列出您要为该资源收集的指标。这些 `measurement` 部分可能指定了完整的指标名称（如 `swap_used`），也可能仅指定将附加到资源类型的指标名称部分。例如，如果在 `diskio` 部分的 `measurement` 部分中指定 `reads`，则会导致收集 `diskio_reads` 指标。

- `collectd` – 可选。指定要使用 `collectd` 协议检索自定义指标。您可以使用 `collectd` 软件将指标发送到 CloudWatch 代理。有关可用于 collectd 的配置选项的更多信息，请参阅 使用 collectd 检索自定义指标。
- `cpu` – 可选。指定要收集的 CPU 指标。该部分仅适用于 Linux 实例。您必须对任何要收集的 CPU 指标至少包含 `resources` 和 `totalcpu` 字段之一。此部分包含以下字段：
  - `drop_original_metrics` – 可选。如果您使用 `metrics` 部分中的 `aggregation_dimensions` 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发

送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

- `resources` – 可选。利用值 * 指定此字段会使系统收集每个 CPU 的指标。唯一允许的值为 *。
- `totalcpu` – 可选。指定是否报告所有 CPU 内核的汇总 CPU 指标。默认值为 true。
- `measurement` – 指定一组要收集的 CPU 指标。可能的值为 `time_active`、`time_guest`、`time_guest_nice`、`time_idle`、`time_iowait`、`time_irq`、`tim` 和 `usage_user`。如果包含 `cpu`，则该字段是必需的。

  默认情况下，`cpu_usage_*` 指标的单位是 `Percent`，`cpu_time_*` 指标没有单位。

  在每个指标的条目中，您可以选择指定下面的一个或两个字段：

  - `rename` – 为该指标指定不同的名称。
  - `unit` – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 `Unit` 描述中所示。
- `metrics_collection_interval` – 可选。指定收集 CPU 指标的频率，从而覆盖配置文件的 `agent` 部分中指定的全局 `metrics_collection_interval`。

  该值是以秒为单位指定的。例如，指定 10 将导致每 10 秒收集一次指标；将其设置为 300，将指定每 5 分钟收集一次指标。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。
- `append_dimensions` – 可选。仅用于 CPU 指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的全局 `append_dimensions` 字段中指定的维度以外，还会使用该字段中的维度。

- `disk` – 可选。指定要收集的磁盘指标。仅收集已装入卷的指标。该部分仅适用于 Linux 实例。此部分包含以下字段：
  - `drop_original_metrics` – 可选。如果您使用 `metrics` 部分中的 `aggregation_dimensions` 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。
  - `resources` – 可选。指定一组磁盘装载点。该字段将 CloudWatch 限制为仅从列出的挂载点中收集指标。您可以指定 * 值以从所有装载点中收集指标。默认值为从所有装载点中收集指标。

- measurement – 指定一组要收集的磁盘指标。可能的值为 free、total、used、used_percent、inodes_free、inodes_used 和 inodes_total。如果包含 disk，则该字段是必需的。

  > **Note**
  >
  > disk 指标具有一个用于 Partition 的维度 ，这意味着生成的自定义指标的数量取决于与您的实例关联的分区数量。您拥有的磁盘分区数取决于您使用的 AMI 以及您附加到服务器的 Amazon EBS 卷数。

  要查看每个 disk 指标的默认单位，请参阅Linux 和 macOS 实例上的 CloudWatch 代理收集的指标。

  在每个指标的条目中，您可以选择指定下面的一个或两个字段：

  - rename – 为该指标指定不同的名称。
  - unit – 指定用于该指标的单位，从而覆盖该指标的 None 默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 Unit 描述中所示。
- ignore_file_system_types – 指定在收集磁盘指标时要排除的文件系统类型。有效值包括 sysfs、devtmpfs 等等。
- drop_device – 将其设置为 true 会导致 Device 不会作为磁盘指标的维度被包括在内。

  对于使用 Nitro 系统的实例来说，防止 Device 被用作维度非常有用，因为在这些实例上，当实例重新启动时，每个磁盘装载的设备名称都会发生变化。这可能会导致指标中的数据不一致，并导致基于这些指标的警报变成 INSUFFICIENT DATA 状态。

  默认为 false。
- metrics_collection_interval – 可选。指定收集磁盘指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

  该值是以秒为单位指定的。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关更多信息，请参阅高精度指标。
- append_dimensions – 可选。指定仅用作磁盘指标附加维度的键值对。如果指定该字段，除了用于该代理收集的所有类型的指标的 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。

您可以使用的一个键值对如下。您还可以指定其他自定义键值对。

- `"VolumeId":"${aws:VolumeId}"` 为您的块设备磁盘指标添加了 `VolumeId` 维度。对于 Amazon EBS 卷，此项将为 Amazon EBS 卷 ID。对于 EC2 实例存储，此项将为设备序列号。使用此项需要将 `drop_device` 参数设置为 `false`。

- `diskio` – 可选。指定要收集的磁盘 I/O 指标。该部分仅适用于 Linux 实例。此部分包含以下字段：
  - `drop_original_metrics` – 可选。如果您使用 `metrics` 部分中的 `aggregation_dimensions` 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。
  - `resources` – 可选。如果指定一组设备，CloudWatch 仅从这些设备中收集指标。否则，将收集所有设备的指标。也可以指定 * 值以从所有设备中收集指标。
  - `measurement` – 指定一组要收集的 diskio 指标。可能的值为 `reads`、`writes`、`read_bytes`、`write_bytes`、`read_time`、`write_time`、`io_time` 和 `iops_in_progress`。如果包含 `diskio`，则该字段是必需的。

    在每个指标的条目中，您可以选择指定下面的一个或两个字段：
    - `rename` – 为该指标指定不同的名称。
    - `unit` – 指定用于该指标的单位，从而覆盖该指标的 None 默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 `Unit` 描述中所示。
  - `metrics_collection_interval` – 可选。指定收集磁盘 I/O 指标的频率，从而覆盖配置文件的 `agent` 部分中指定的全局 `metrics_collection_interval`。

    该值是以秒为单位指定的。

    如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。
  - `append_dimensions` – 可选。仅用于磁盘 I/O 指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的 `append_dimensions` 字段中指定的维度以外，还会使用该字段中的维度。

- `swap` – 可选。指定要收集的交换内存指标。该部分仅适用于 Linux 实例。此部分包含以下字段：
  - `drop_original_metrics` – 可选。如果您使用 `metrics` 部分中的 `aggregation_dimensions` 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到

CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

- `measurement` – 指定一组要收集的交换内存指标。可能的值为 `free`、`used` 和 `used_percent`。如果包含 `swap`，则该字段是必需的。

  要查看每个 `swap` 指标的默认单位，请参阅Linux 和 macOS 实例上的 CloudWatch 代理收集的指标。

  在每个指标的条目中，您可以选择指定下面的一个或两个字段：

  - `rename` – 为该指标指定不同的名称。
  - `unit` – 指定用于该指标的单位，从而覆盖该指标的 `None` 默认单位 `None`。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 `Unit` 描述中所示。
- `metrics_collection_interval` – 可选。指定收集交换内存指标的频率，从而覆盖配置文件的 `agent` 部分中指定的全局 `metrics_collection_interval`。

  该值是以秒为单位指定的。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。
- `append_dimensions` – 可选。仅用于交换内存指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的全局 `append_dimensions` 字段中指定的维度以外，还会使用该字段中的维度。这是作为高精度指标收集的。

- `mem` – 可选。指定要收集的内存指标。该部分仅适用于 Linux 实例。此部分包含以下字段：
  - `drop_original_metrics` – 可选。如果您使用 `metrics` 部分中的 `aggregation_dimensions` 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。
  - `measurement` – 指定一组要收集的内存指标。可能的值为 `active`、`available`、`available_percent`、`buffered`、`cached`、`free`、`inactive`、`total`、`u` 和 `used_percent`。如果包含 `mem`，则该字段是必需的。

    要查看每个 `mem` 指标的默认单位，请参阅Linux 和 macOS 实例上的 CloudWatch 代理收集的指标。

    在每个指标的条目中，您可以选择指定下面的一个或两个字段：

    - `rename` – 为该指标指定不同的名称。

  - unit – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 Unit 描述中所示。
- metrics_collection_interval – 可选。指定收集内存指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

  该值是以秒为单位指定的。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。
- append_dimensions – 可选。仅用于内存指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。

- net – 可选。指定要收集的每个联网指标。该部分仅适用于 Linux 实例。此部分包含以下字段：
  - drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。
  - resources – 可选。如果指定一组网络接口，CloudWatch 仅从这些接口中收集指标。否则，将收集所有设备的指标。也可以指定 * 值以从所有接口中收集指标。
  - measurement – 指定一组要收集的联网指标。可能的值为 bytes_sent、bytes_recv、drop_in、drop_out、err_in、err_out、packets_sent 和 packets_recv。如果包含 net，则该字段是必需的。

    要查看每个 net 指标的默认单位，请参阅Linux 和 macOS 实例上的 CloudWatch 代理收集的指标。

    在每个指标的条目中，您可以选择指定下面的一个或两个字段：
    - rename – 为该指标指定不同的名称。
    - unit – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 Unit 描述中所示。
  - metrics_collection_interval – 可选。指定收集网络指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

    该值是以秒为单位指定的。例如，指定 10 将导致每 10 秒收集一次指标；将其设置为 300，将指定每 5 分钟收集一次指标。

如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。

- append_dimensions – 可选。仅用于网络指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。

- netstat – 可选。指定收集 TCP 连接状态和 UDP 连接指标。该部分仅适用于 Linux 实例。此部分包含以下字段：

  - drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

  - measurement – 指定一组要收集的网络状态指标。可能的值为 tcp_close、tcp_close_wait、tcp_closing、tcp_established、tcp_fin_wait1、tcp_fi
和 udp_socket。如果包含 netstat，则该字段是必需的。

    要查看每个 netstat 指标的默认单位，请参阅Linux 和 macOS 实例上的 CloudWatch 代理收集的指标。

    在每个指标的条目中，您可以选择指定下面的一个或两个字段：

    - rename – 为该指标指定不同的名称。
    - unit – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 Unit 描述中所示。

  - metrics_collection_interval – 可选。指定收集网络状态指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

    该值是以秒为单位指定的。

    如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关高精度指标的更多信息，请参阅 高精度指标。

  - append_dimensions – 可选。仅用于网络状态指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。

- processes – 可选。指定要收集的进程指标。该部分仅适用于 Linux 实例。此部分包含以下字段：

- drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。
- measurement – 指定一组要收集的进程指标。可能的值为 blocked、dead、idle、paging、running、sleeping、stopped、total、total_threads、w 和 zombies。如果包含 processes，则该字段是必需的。

  对于所有 processes 指标，默认单位为 None。

  在每个指标的条目中，您可以选择指定下面的一个或两个字段：

  - rename – 为该指标指定不同的名称。
  - unit – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 Unit 描述中所示。
- metrics_collection_interval – 可选。指定收集进程指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

  该值是以秒为单位指定的。例如，指定 10 将导致每 10 秒收集一次指标；将其设置为 300，将指定每 5 分钟收集一次指标。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关更多信息，请参阅高精度指标。
- append_dimensions – 可选。仅用于进程指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。

- nvidia_gpu – 可选。指定要收集的 NVIDIA GPU 指标。本部分仅适用于配置了 NVIDIA GPU 加速器并安装了 NVIDIA 系统管理界面（nvidia-smi）的主机上的 Linux 实例。

  收集的 NVIDIA GPU 指标以字符串 nvidia_smi_ 作为前缀以与收集的其他加速器类型的指标进行区分。此部分包含以下字段：

  - drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

  - `measurement` – 指定一组要收集的 NVIDIA GPU 指标。有关此处可能使用的值的列表，请参阅 收集 NVIDIA GPU 指标 表中的指标列。

    在每个指标的条目中，您可以选择指定以下一个或两个字段:

    - `rename` – 为该指标指定不同的名称。
    - `unit` – 指定用于该指标的单位，从而覆盖该指标的默认单位 None。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 `Unit` 描述中所示。
  - `metrics_collection_interval` – 可选。指定收集 NVIDIA GPU 指标的频率，从而覆盖配置文件的 `agent` 部分中指定的全局 `metrics_collection_interval`。
- `procstat` – 可选。指定您希望从各个流程检索指标。有关可用于 procstat 的配置选项的更多信息，请参阅 使用 procstat 插件收集流程指标。
- `statsd` – 可选。指定要使用 `StatsD` 协议检索自定义指标。CloudWatch 代理充当协议的守护进程。您可以使用任何标准 `StatsD` 客户端将指标发送到 CloudWatch 代理。有关可用于 StatsD 的配置选项的更多信息，请参阅 使用 StatsD 检索自定义指标 。
- `ethtool` – 可选。指定要使用 `ethtool` 插件检索网络指标。此插件可以导入标准 ethtool 实用工具收集的指标，以及 Amazon EC2 实例中的网络性能指标。有关可用于 ethtool 的配置选项的更多信息，请参阅 收集网络性能指标。

下面是一个适用于 Linux 服务器的 `metrics` 部分示例。在此示例中，会收集三个 CPU 指标、三个 netstat 指标、三个流程指标和一个磁盘指标，并将代理设置为从 `collectd` 客户端接收其他指标。

```
"metrics": {
    "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId",
 "InstanceType"],[]],
    "metrics_collected": {
      "collectd": {},
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "drop_original_metrics": [ "cpu_usage_guest" ],
        "metrics_collection_interval": 10,
```

```
        "append_dimensions": {
          "test": "test1",
          "date": "2017-10-01"
        }
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_syn_sent",
          "tcp_close"
        ],
        "metrics_collection_interval": 60
      },
       "disk": {
        "measurement": [
          "used_percent"
        ],
        "resources": [
          "*"
        ],
        "drop_device": true
      },
      "processes": {
        "measurement": [
          "running",
          "sleeping",
          "dead"
        ]
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    }
  }
```

Windows Server

在 Windows Server 的 `metrics_collected` 部分中，您可以包含每个 Windows 性能对象的小节，例如，`Memory`、`Processor` 和 `LogicalDisk`。有关可用的对象和计数器的信息，请参阅 Microsoft Windows 文档性能计数器。

在每个对象的小节中，您可以指定要收集的计数器的 measurement 数组。需要为在配置文件中指定的每个对象指定 measurement 数组。您也可以指定 resources 字段以命名从中收集指标的实例。您还可以为 resources 指定 *，以便为每个实例收集单独的指标。如果省略具有实例的计数器的 resources，则所有实例的数据会聚合到一个组中。如果省略没有实例的计数器的 resources，CloudWatch 代理则不会收集计数器。要确定计数器是否有实例，可以使用以下命令之一。

Powershell：

```
Get-Counter -ListSet *
```

命令行（不是 Powershell）：

```
TypePerf.exe -q
```

在每个对象部分中，您还可以指定以下可选字段：

- metrics_collection_interval – 可选。指定为该对象收集指标的频率，从而覆盖配置文件的 agent 部分中指定的全局 metrics_collection_interval。

  该值是以秒为单位指定的。例如，指定 10 将导致每 10 秒收集一次指标；将其设置为 300，将指定每 5 分钟收集一次指标。

  如果您将该值设置为低于 60 秒，则将每个指标作为高精度指标进行收集。有关更多信息，请参阅 高精度指标。
- append_dimensions – 可选。指定仅用于该对象的指标的其他维度。如果指定该字段，除了用于该代理收集的所有类型的指标的全局 append_dimensions 字段中指定的维度以外，还会使用该字段中的维度。
- drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

在每个计数器部分中，您还可以指定以下可选字段：

- rename – 指定在 CloudWatch 中用于该指标的不同名称。

- `unit` – 指定用于该指标的单位。您指定的单位必须是有效的 CloudWatch 指标单位，如 MetricDatum 的 `Unit` 描述中所示。

您可以在 `metrics_collected` 中包括两个其他可部分：

- `statsd` – 让您能够使用 `StatsD` 协议检索自定义指标。CloudWatch 代理充当协议的守护进程。您可以使用任何标准 `StatsD` 客户端将指标发送到 CloudWatch 代理。有关更多信息，请参阅 使用 StatsD 检索自定义指标 。
- `procstat` – 让您能够从各个流程检索指标。有关更多信息，请参阅 使用 procstat 插件收集流程指标。

下面是一个用于 Windows Server 的 `metrics` 部分的示例。在此示例中，会收集许多 Windows 指标，并且计算机也被设置为从 `StatsD` 客户端接收其他指标。

```
"metrics": {
    "metrics_collected": {
      "statsd": {},
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
```

```
        ]
      },
      "Memory": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Available Bytes",
          "Cache Faults/sec",
          "Page Faults/sec",
          "Pages/sec"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "Network Interface": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Bytes Received/sec",
          "Bytes Sent/sec",
          "Packets Received/sec",
          "Packets Sent/sec"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "System": {
        "measurement": [
          "Context Switches/sec",
          "System Calls/sec",
          "Processor Queue Length"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
```

```
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"], ["d1"],[]]
    }
  }
```

CloudWatch 代理配置文件：Logs（日志）部分

logs 部分包含以下字段：

- logs_collected – 如果包含 logs 部分，则为必需。指定从服务器中收集的日志文件和 Windows 事件日志。它可以包含两个字段：files 和 windows_events。
  - files – 指定 CloudWatch 代理收集的常规日志文件。它包含一个字段 (collect_list)，它进一步定义这些文件。
    - collect_list – 如果包含 files，则为必需。包含一组条目，每个条目指定一个要收集的日志文件。其中的每个条目可以包含以下字段：
      - file_path – 指定要上传到 CloudWatch Logs 的日志文件的路径。接受标准 Unix Glob 匹配规则，并添加 ** 以作为超级星号。例如，指定 /var/log/**.log 将导致收集 /var/log 目录树中的所有 .log 文件。有关更多示例，请参阅 Glob 库。

        您也可以使用标准星号作为标准通配符。例如，/var/log/system.log* 与 /var/log 中的 system.log_1111、system.log_2222 等文件匹配。

        根据文件修改时间，只有最新文件才会推送到 CloudWatch Logs。我们建议您使用通配符指定同一类型的一系列文件（如 access_log.2018-06-01-01 和 access_log.2018-06-01-02）而不是多个不同类型的文件（如 access_log_80 和 access_log_443）。要指定多个类型的文件，请向代理配置文件再添加一个日志流条目，让每种日志文件都转到不同的日志流。
      - auto_removal – 可选。如果该值为 true，且此日志文件已经轮换，则 CloudWatch 代理会在读取到该文件后自动将其删除。通常，日志文件将其在全部内容上传到 CloudWatch Logs 后被删除，但如果代理到达 EOF（文件末尾）并同时检测到其他具有相同 file_path 的新日志文件，则代理将删除旧文件，因此，您必须确保您已在创建新文件前完成对旧文件的写入。RUST 跟踪库存在已知的不兼容性，因为它可能会在创建一个新的日志文件后，仍尝试写入旧的日志文件。

        代理仅从创建多个文件的日志中删除完整的文件，例如为每个日期创建单独文件的日志。如果日志连续写入单个文件，则不会删除该日志。

如果您已经有轮换或删除日志文件的方法，建议您忽略此字段或者将其设置为 false。

如果省略该字段，则使用 false 的默认值。

- log_group_name – 可选。指定要在 CloudWatch Logs 中使用的日志组名称。

  我们建议您使用此字段指定日志组，以防混淆。如果您省略 log_group_name，系统会将最后一个点之前的 file_path 的值作为日志组名称。例如，如果文件路径为 /tmp/TestLogFile.log.2017-07-11-14，则日志组名称为 /tmp/TestLogFile.log。

  如果您指定日志组名称，您可以使用 {instance_id}、{hostname}、{local_hostname} 和 {ip_address} 作为名称中的变量。{hostname} 会从 EC2 元数据中检索主机名，{local_hostname} 会使用网络配置文件中的主机名。

  如果使用这些变量创建许多不同的日志组，请记住每个区域每个账户存在 1000000 个日志组的限制。

  允许的字符包括 a–z、A–Z、0–9、“_”（下划线）、“-”（连字符）、“/”（正斜杠）和“.”（句点）。

- log_group_class – 可选。指定要用于新日志组的日志组类。有关日志组类的更多信息，请参阅 Log classes。

  有效值为 STANDARD 和 INFREQUENT_ACCESS。如果省略该字段，则使用 STANDARD 的默认值。

  > **Important**
  >
  > 在创建日志组后，无法更改其类。

- log_stream_name – 可选。指定要在 CloudWatch Logs 中使用的日志流名称。您可以在名称中包含 {instance_id}、{hostname}、{local_hostname} 和 {ip_address} 作为变量。{hostname} 从 EC2 元数据中检索主机名，{local_hostname} 使用网络配置文件中的主机名。

  如果省略此字段，则使用全局 logs 部分中 log_stream_name 参数的值。如果也省略了该值，则使用 {instance_id} 的默认值。

  如果还没有日志流，则会自动创建一个日志流。

- `retention_in_days` – 可选。在指定的日志组中指定保留日志事件的天数。
  - 如果代理目前正在创建此日志组，并且您忽略此字段，则此新日志组的保留期将设置为永不过期。
  - 如果此日志组已存在，并且您指定了此字段，则将使用您指定的新保留期。如果您为已存在的日志组忽略此字段，则日志组的保留不会更改。

    CloudWatch 代理向导在创建代理配置文件且您未指定日志保留值时，使用 `-1` 作为此字段的默认值。此 `-1` 值由向导设置，其指定日志组中的事件永不过期。但是，手动将此值编辑为 `-1` 是无效的。

  有效的值为
  1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、2192、2557、29
  和 3653。

  如果配置代理以将多个日志流写入同一个日志组，则在一个位置指定 `retention_in_days` 将为整个日志组设置日志保留。如果您在多个位置为同一日志组指定 `retention_in_days`，则在所有这些值均相等时才会设置保留。但是，如果在多个位置为同一日志组指定不同的 `retention_in_days` 值，则不会设置日志保留，代理将停止并返回错误。

  > **Note**
  >
  > 代理的 IAM 角色或 IAM 用户必须拥有 `logs:PutRetentionPolicy` 才能设置保留策略。有关更多信息，请参阅 允许 CloudWatch 代理设置日志保留策略。

  > **Warning**
  >
  > 如果您为已存在的日志组设置 `retention_in_days`，则该日志组中在您指定天数之前发布的所有日志都将被删除。例如，将其设置为 3 会导致 3 天前和之前的所有日志被删除。

- `filters` – 可选。可以包含一组条目，其中的每个条目都指定一个正则表达式和筛选条件类型以指定是否发布或删除与筛选条件匹配的日志条目。如果您忽略此字段，则日志文件中的所有日志都会发布到 CloudWatch Logs。如果您包括此字段，则代理将使用您指定的所有筛选条件处理每条日志消息，并且只有通过所有筛选条件的日志事件才会发布到

CloudWatch Logs。未通过所有筛选条件的日志条目仍将保留在主机的日志文件中，但不会发送到 CloudWatch Logs。

筛选条件数组中的每个条目可以包含以下字段:

- `type`— 表示筛选条件类型。有效值为 `include` 和 `exclude`。对于 `include`，日志条目必须与要发布到 CloudWatch Logs 的表达式匹配。对于 `exclude`，与筛选条件匹配的每个日志条目不会发送到 CloudWatch Logs。
- `expression`— 遵循 RE2 语法的正则表达式字符串。

> **Note**
>
> CloudWatch 代理不会检查您提供的任何正则表达式的性能，也不会限制正则表达式评估的运行时间。我们建议您不要编写评估昂贵的表达式。有关可能问题的更多信息，请参阅正则表达式拒绝服务 - ReDoS

例如，以下 CloudWatch 代理配置文件的摘录会向 CloudWatch Logs 发布 PUT 和 POST 请求的日志，但不包括来自 Firefox 的日志。

```
"collect_list": [
  {
    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
    "log_group_name": "test.log",
    "log_stream_name": "test.log",
    "filters": [
      {
        "type": "exclude",
        "expression": "Firefox"
      },
      {
        "type": "include",
        "expression": "P(UT|OST)"
      }
    ]
  },
  .....
]
```

> **Note**
>
> 配置文件中筛选条件的顺序对性能至关重要。在上一个示例中，代理在开始评估第二个筛选条件之前会删除与 Firefox 匹配的所有日志。为了减少多个筛选条件评估的日志条目，首先将预计排除更多日志的筛选条件放在配置文件中。

- timezone – 可选。指定在日志事件中放置时间戳时要使用的时区。有效值为 UTC 和 Local。默认值为 Local。

  如果您不指定 timestamp_format 的值，此参数将被忽略。

- timestamp_format – 可选。使用纯文本和以 % 开头的特殊符号指定时间戳格式。如果省略该字段，则使用当前时间。如果使用该字段，您可以使用以下列表中的符号作为格式的一部分。

  如果单个日志条目包含两个与该格式相符的时间戳，则使用第一个时间戳。

  此符号列表与较旧的 CloudWatch Logs 代理使用的列表不同。有关这些区别的摘要，请参阅统一的 CloudWatch 代理和更早的 CloudWatch Logs 代理之间的时间戳差异。

  %y

  使用以零填充的十进制数字形式的没有世纪的年份。例如，19 表示 2019 年。

  %Y

  使用十进制数字形式的具有世纪的年份。例如，2019。

  %b

  使用区域设置的缩写名称形式的月份

  %B

  使用区域设置的完整名称形式的月份

  %m

  使用以零填充的十进制数字形式的月份

  %-m

  十进制数字形式的月份 (不使用零填充)

%d

使用以零填充的十进制数字形式的日期

%-d

十进制数字形式的日期 (不使用零填充)

%A

星期的全称，例如 Monday

%a

星期的缩写，例如 Mon

%H

使用以零填充的十进制数字形式的小时 (24 小时制)

%I

使用以零填充的十进制数字形式的小时 (12 小时制)

%-I

十进制数字形式的小时（12 小时制）（不使用零填充）

%p

AM 或 PM

%M

使用以零填充的十进制数字形式的分钟

%-M

十进制数字形式的分钟 (不使用零填充)

%S

使用以零填充的十进制数字形式的秒

%-S

十进制数字形式的秒 (不使用零填充)

%f

小数秒采用十进制数（1-9 位数），左侧用零填充。

%Z

时区，例如 PST

%z

时区，以本地时区与 UTC 的偏移量表示。例如，-0700。仅支持此格式。例如，-07:00 不是有效格式。

- multi_line_start_pattern – 指定用于识别日志消息开头的模式。日志消息由与模式匹配的行以及与模式不匹配的任何后续行组成。

  如果您省略此字段，则会禁用多行模式，而且以非空格字符开头的任何行都会使上一个日志消息结束 (如果存在)，并开始新的日志消息。

  如果包含该字段，您可以指定 {timestamp_format} 以使用与您的时间戳格式相同的正则表达式。否则，您可以为 CloudWatch Logs 指定不同的正则表达式，以用于确定多行条目的起始行。

- encoding – 指定日志文件的编码，以便正确读取该文件。如果您指定错误的编码，则可能导致数据丢失，因为无法解码的字符将被其他字符替代。

  默认值为 utf-8。以下是所有可能值：

  ascii, big5, euc-jp, euc-kr, gbk, gb18030, ibm866, iso2022-jp, iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8, utf-16, utf-16le, UTF-16, UTF-16LE, windows-874, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, x-mac-cyrillic

- windows_events 部分指定从运行 Windows Server 的服务器中收集的 Windows 事件类型。其中包括以下字段：
  - collect_list – 如果包含 windows_events，则为必需。指定要收集的 Windows 事件的类型和级别。要收集的每个日志在该部分中包含一个条目，其中可以包含以下字段：

- `event_name` – 指定要记录的 Windows 事件的类型。这等同于 Windows 事件日志通道名称：例如 `System`、`Security`、`Application` 等。要记录的每种类型的 Windows 事件需要使用该字段。

  > **Note**
  >
  > 当 CloudWatch 从 Windows 日志通道检索消息时，它会根据其 `Full Name` 属性查找日志通道。同时，Windows 事件查看器导航窗格将显示日志通道的 `Log Name` 属性。`Full Name` 和 `Log Name` 并不总是匹配。确认通道的 `Full Name`，在 Windows 事件查看器中右键单击，然后打开 Properties（属性）。

- `event_levels` – 指定要录入的事件的级别。您必须指定要录入的每个级别。可能的值包括 `INFORMATION`、`WARNING`、`ERROR`、`CRITICAL` 和 `VERBOSE`。要记录的每种类型的 Windows 事件需要使用该字段。
- `log_group_name` – 必需。指定要在 CloudWatch Logs 中使用的日志组名称。
- `log_stream_name` – 可选。指定要在 CloudWatch Logs 中使用的日志流名称。您可以在名称中包含 `{instance_id}`、`{hostname}`、`{local_hostname}` 和 `{ip_address}` 作为变量。`{hostname}` 从 EC2 元数据中检索主机名，`{local_hostname}` 使用网络配置文件中的主机名。

  如果省略此字段，则使用全局 `logs` 部分中 `log_stream_name` 参数的值。如果也省略了该值，则使用 `{instance_id}` 的默认值。

  如果还没有日志流，则会自动创建一个日志流。
- `event_format` – 可选。指定在 CloudWatch Logs 中存储 Windows 事件时要使用的格式。与在 Windows 事件查看器中一样，`xml` 使用 XML 格式。`text` 使用原 CloudWatch Logs 代理格式。
- `retention_in_days` – 可选。在指定的日志组中指定保留 Windows 事件的天数。
  - 如果代理目前正在创建此日志组，并且您忽略此字段，则此新日志组的保留期将设置为永不过期。
  - 如果此日志组已存在，并且您指定了此字段，则将使用您指定的新保留期。如果您为已存在的日志组忽略此字段，则日志组的保留不会更改。

    CloudWatch 代理向导在创建代理配置文件且您未指定日志保留值时，使用 `-1` 作为此字段的默认值。此 `-1` 值由向导设置，指定日志组中的事件不会过期。但是，手动将此值编辑为 `-1` 是无效的。

有效的值为
1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、2192、2557、29
和 3653。

如果配置代理以将多个日志流写入同一个日志组，则在一个位置指定 `retention_in_days` 将为整个日志组设置日志保留。如果您在多个位置为同一日志组指定 `retention_in_days`，则在所有这些值均相等时才会设置保留。但是，如果在多个位置为同一日志组指定不同的 `retention_in_days` 值，则不会设置日志保留，代理将停止并返回错误。

> **Note**
>
> 代理的 IAM 角色或 IAM 用户必须拥有 `logs:PutRetentionPolicy` 才能设置保留策略。有关更多信息，请参阅 允许 CloudWatch 代理设置日志保留策略。

> **Warning**
>
> 如果您为已存在的日志组设置 `retention_in_days`，则该日志组中在您指定天数之前发布的所有日志都将被删除。例如，将其设置为 3 会导致 3 天前和之前的所有日志被删除。

- `log_stream_name` – 必需。对于未在 `collect_list` 的相应条目内的 `log_stream_name` 参数中定义各日志流名称的任何日志或 Windows 事件，指定要用于它们的默认日志流名称。
- `endpoint_override` – 指定一个 FIPS 端点或私有链接，以用作代理在其中发送日志的端点。如果指定此字段并设置私有链接，您可以将日志发送到 Amazon VPC 终端节点。有关更多信息，请参阅 Amazon VPC 是什么？。

  `endpoint_override` 的值必须是表示 URL 的字符串。

  例如，在配置文件的日志部分中，以下内容将代理设置为在发送日志时使用 VPC 终端节点。

```
{
  "logs": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXX.logs.us-
east-1.vpce.amazonaws.com",
   ......
   },
```

```
}
```

- `force_flush_interval` – 以秒为单位指定日志在发送到服务器之前保留在内存缓冲区中的最大时间量。无论此字段的设置如何，如果缓冲区中的日志大小达到 1 MB，日志会立即发送到服务器。默认值是 5。

  如果您使用代理以嵌入式指标格式报告高分辨率指标，并且正在为这些指标设置警报，请将此参数设置为默认值 5。否则，报告指标时会出现延迟，这可能会导致出现警报，告知存在部分或不完整的数据。

- `credentials` – 指定一个在向不同 AWS 账户发送日志时使用的 IAM 角色。如果指定，则此字段包含一个参数 `role_arn`。
  - `role_arn` – 指定一个在向不同 AWS 账户发送日志时用于身份验证的 IAM 角色的 ARN。有关更多信息，请参阅 向不同账户发送指标、日志和跟踪信息。如果在此指定，它将覆盖在配置文件的 `agent` 部分指定的 `role_arn`（如果有）。
- `metrics_collected`：此字段可包含特定部分，用于指定代理将收集日志以启用使用案例，例如 CloudWatch Application Signals 和针对 Amazon EKS 增强了可观测性的 Container Insights。
  - `application_signals`（可选）指定您要启用 CloudWatch Application Signals。有关此配置的更多信息，请参阅 启用 CloudWatch Application Signals。
  - `kubernetes` – 此字段可以包含 `enhanced_container_insights` 参数，用于启用针对 Amazon EKS 增强了可观测性的 Container Insights。
    - `enhanced_container_insights` – 将其设置为 `true`，可启用针对 Amazon EKS 增强了可观测性的 Container Insights。有关更多信息，请参阅 针对 Amazon EKS 增强了可观测性的 Container Insights。
    - `accelerated_compute_metrics` – 将其设置为 `false`，从而选择不收集 Amazon EKS 集群上的 Nvidia GPU 指标。有关更多信息，请参阅 NVIDIA GPU 指标。
  - `emf` – 要收集日志中嵌入的指标，不再需要添加此 `emf` 字段。这是旧字段，指定代理将收集采用嵌入式指标格式的日志。您可以从这些日志生成指标数据。有关更多信息，请参阅 在日志中嵌入指标。

下面是一个 `logs` 部分示例。

```
"logs":
   {
       "logs_collected": {
           "files": {
               "collect_list": [
```

```
                    {
                        "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\amazon-cloudwatch-agent.log",
                        "log_group_name": "amazon-cloudwatch-agent.log",
                        "log_stream_name": "my_log_stream_name_1",
                        "timestamp_format": "%H: %M: %S%y%b%-d"
                    },
                    {
                        "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\test.log",
                        "log_group_name": "test.log",
                        "log_stream_name": "my_log_stream_name_2"
                    }
                ]
            },
            "windows_events": {
                "collect_list": [
                    {
                        "event_name": "System",
                        "event_levels": [
                            "INFORMATION",
                            "ERROR"
                        ],
                        "log_group_name": "System",
                        "log_stream_name": "System"
                    },
                    {
                        "event_name": "CustomizedName",
                        "event_levels": [
                            "INFORMATION",
                            "ERROR"
                        ],
                        "log_group_name": "CustomizedLogGroup",
                        "log_stream_name": "CustomizedLogStream"
                    }
                ]
            }
        },
        "log_stream_name": "my_log_stream_name",
        "metrics_collected": {
            "kubernetes": {
                 "enhanced_container_insights": true
       }
     }
```

```
  }
```

CloudWatch 代理配置文件：跟踪信息部分

通过向 CloudWatch 代理配置文件添加 `traces` 部分，您可以启用 CloudWatch Application Sights，或从 X-Ray 和 OpenTelemetry 检测开发工具包收集跟踪信息，然后将其发送到 X-Ray。

> **Important**
>
> 代理的 IAM 角色或 IAM 用户必须具有 AWSXrayWriteOnlyAccess 策略才能将跟踪数据发送到 X-Ray。有关更多信息，请参阅 创建 IAM 角色和用户以用于 CloudWatch 代理。

要快速开始收集跟踪信息，您只需将以下内容添加到 CloudWatch 代理配置文件中即可。

```
"traces_collected": {
        "xray": {
        },
        "otlp": {
        }
      }
```

如果将上一部分添加到 CloudWatch 代理配置文件，然后重新启动代理，这将使代理开始使用以下默认选项和值收集跟踪。有关这些参数的更多信息，请参阅本节后面的参数定义。

```
"traces_collected": {
        "xray": {
            "bind_address": "127.0.0.1:2000",
            "tcp_proxy": {
              "bind_address": "127.0.0.1:2000"
            }
        },
        "otlp": {
            "grpc_endpoint": "127.0.0.1:4317",
            "http_endpoint": "127.0.0.1:4318"
        }
      }
```

`traces` 部分可以包括以下字段：

- `traces_collected` – 如果包含 `traces` 部分，则为必需。指定要从哪些 SDK 收集跟踪信息。它可以包括以下字段：
  - `application_signals` – 可选。指定您要启用 CloudWatch Application Signals。有关此配置的更多信息，请参阅 启用 CloudWatch Application Signals。
  - `xray` – 可选。指定您要从 X-Ray SDK 收集跟踪信息。此部分包含以下字段：
    - `bind_address` – 可选。指定 CloudWatch 代理用于侦听 X-Ray 跟踪信息的 UDP 地址。格式为 `ip:port`。此地址必须与 X-Ray SDK 中设置的地址相匹配。

      如果省略该字段，则使用 `127.0.0.1:2000` 的默认值。
    - `tcp_proxy` – 可选。为用于支持 X-Ray 远程采样的代理配置地址。有关更多信息，请参阅 X-Ray 文档中的配置采样规则。

      此部分可以包含以下字段。
      - `bind_address` – 可选。指定 CloudWatch 代理应将代理设置到的 TCP 地址。格式为 `ip:port`。此地址必须与 X-Ray SDK 中设置的地址相匹配。

        如果省略该字段，则使用 `127.0.0.1:2000` 的默认值。
  - `otlp` – 可选。指定您要从 OpenTelemetry SDK 收集跟踪信息。有关 AWS Distro for OpenTelemetry 的更多信息，请参阅 AWS Distro for OpenTelemetry。有关适用于 OpenTelemetry SDK 的 AWS Distro 的更多信息，请参阅简介。

    此部分包含以下字段：
    - `grpc_endpoint` – 可选。指定 CloudWatch 代理用于侦听使用 gRPC 远程过程调用发送的 OpenTelemetry 跟踪信息的地址。格式为 `ip:port`。此地址必须与 OpenTelemetry SDK 中为 gRPC 导出器设置的地址相匹配。

      如果省略该字段，则使用 `127.0.0.1:4317` 的默认值。
    - `http_endpoint` – 可选。指定 CloudWatch 代理用于侦听通过 HTTP 发送的 OTLP 跟踪信息的地址。格式为 `ip:port`。此地址必须与 OpenTelemetry SDK 中为 HTTP 导出器设置的地址相匹配。

      如果省略该字段，则使用 `127.0.0.1:4318` 的默认值。
- `concurrency` – 可选。指定可用于上传跟踪信息的 X-Ray 的最大并发调用次数。默认值为 8
- `local_mode` – 可选。如果为 `true`，则代理不会收集 Amazon EC2 实例元数据。默认值为 `false`

- endpoint_override – 可选。指定一个 FIPS 端点或私有链接，以用作 CloudWatch 代理在其中发送跟踪信息的端点。指定此字段并设置私有链接，使您能够将跟踪信息发送到 Amazon VPC 端点。有关更多信息，请参阅 Amazon VPC 是什么

  endpoint_override 的值必须是表示 URL 的字符串。
- region_override – 可选。指定用于 X-Ray 端点的区域。CloudWatch 代理会将跟踪信息发送到指定区域中的 X-Ray。如果省略此字段，该代理会将跟踪信息发送到 Amazon EC2 实例所在的区域。

  如果您在此处指定区域，则此区域将优先于配置文件 agent 部分中 region 参数的设置。
- proxy_override – 可选。指定 CloudWatch 代理向 X-Ray 发送请求时使用的代理服务器地址。必须将代理服务器的协议指定为此地址的组成部分。
- credentials – 指定一个在向不同 AWS 账户发送跟踪信息时使用的 IAM 角色。如果指定，则此字段包含一个参数 role_arn。
  - role_arn – 指定一个在向不同 AWS 账户发送跟踪信息时用于身份验证的 IAM 角色的 ARN。有关更多信息，请参阅 向不同账户发送指标、日志和跟踪信息。如果在此指定，它将覆盖在配置文件的 agent 部分指定的 role_arn（如果有）。

CloudWatch 代理配置文件：完整示例

下面是一个适用于 Linux 服务器的完整 CloudWatch 代理配置文件示例。

要收集的指标的 measurement 部分中列出的项目可能指定了完整的指标名称，也可能仅指定将附加到资源类型的指标名称部分。例如，如果在 diskio 部分的 measurement 部分中指定 reads 或 diskio_reads，将导致收集 diskio_reads 指标。

该示例提供了两种方法以在 measurement 部分中指定指标。

```
{
  "agent": {
    "metrics_collection_interval": 10,
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
  },
  "metrics": {
    "namespace": "MyCustomNamespace",
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
```

```
            ],
            "measurement": [
              {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit":
 "Percent"},
              {"name": "cpu_usage_nice", "unit": "Percent"},
              "cpu_usage_guest"
            ],
            "totalcpu": false,
            "metrics_collection_interval": 10,
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "disk": {
            "resources": [
              "/",
              "/tmp"
            ],
            "measurement": [
              {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
              "total",
              "used"
            ],
             "ignore_file_system_types": [
              "sysfs", "devtmpfs"
            ],
            "metrics_collection_interval": 60,
            "append_dimensions": {
              "customized_dimension_key_3": "customized_dimension_value_3",
              "customized_dimension_key_4": "customized_dimension_value_4"
            }
          },
          "diskio": {
            "resources": [
              "*"
            ],
            "measurement": [
              "reads",
              "writes",
              "read_time",
              "write_time",
              "io_time"
            ],
```

```
            "metrics_collection_interval": 60
          },
          "swap": {
            "measurement": [
              "swap_used",
              "swap_free",
              "swap_used_percent"
            ]
          },
          "mem": {
            "measurement": [
              "mem_used",
              "mem_cached",
              "mem_total"
            ],
            "metrics_collection_interval": 1
          },
          "net": {
            "resources": [
              "eth0"
            ],
            "measurement": [
              "bytes_sent",
              "bytes_recv",
              "drop_in",
              "drop_out"
            ]
          },
          "netstat": {
            "measurement": [
              "tcp_established",
              "tcp_syn_sent",
              "tcp_close"
            ],
            "metrics_collection_interval": 60
          },
          "processes": {
            "measurement": [
              "running",
              "sleeping",
              "dead"
            ]
          }
        },
```

```
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]],
        "force_flush_interval" : 30
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "log_stream_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                "log_group_name": "test.log",
                "log_stream_name": "test.log",
                "timezone": "Local"
              }
            ]
          }
        },
        "log_stream_name": "my_log_stream_name",
        "force_flush_interval" : 15,
        "metrics_collected": {
           "kubernetes": {
                "enhanced_container_insights": true
      }
    }
  }
}
```

下面是一个适用于运行 Windows Server 的服务器的完整 CloudWatch 代理配置文件示例。

```
{
```

```
      "agent": {
        "metrics_collection_interval": 60,
        "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "Processor": {
            "measurement": [
              {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
              "% Interrupt Time",
              "% User Time",
              "% Processor Time"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "LogicalDisk": {
            "measurement": [
              {"name": "% Idle Time", "unit": "Percent"},
              {"name": "% Disk Read Time", "rename": "DISK_READ"},
              "% Disk Write Time"
            ],
            "resources": [
              "*"
            ]
          },
          "customizedObjectName": {
            "metrics_collection_interval": 60,
            "customizedCounterName": [
              "metric1",
              "metric2"
            ],
            "resources": [
              "customizedInstances"
            ]
          },
          "Memory": {
```

```
            "metrics_collection_interval": 5,
            "measurement": [
              "Available Bytes",
              "Cache Faults/sec",
              "Page Faults/sec",
              "Pages/sec"
            ]
          },
          "Network Interface": {
            "metrics_collection_interval": 5,
            "measurement": [
              "Bytes Received/sec",
              "Bytes Sent/sec",
              "Packets Received/sec",
              "Packets Sent/sec"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_3": "customized_dimension_value_3"
            }
          },
          "System": {
            "measurement": [
              "Context Switches/sec",
              "System Calls/sec",
              "Processor Queue Length"
            ]
          }
        },
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
["d1"],[]]
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
```

```
            {
              "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\amazon-cloudwatch-agent.log",
              "log_group_name": "amazon-cloudwatch-agent.log",
              "timezone": "UTC"
            },
            {
              "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\test.log",
              "log_group_name": "test.log",
              "timezone": "Local"
            }
          ]
        },
        "windows_events": {
          "collect_list": [
            {
              "event_name": "System",
              "event_levels": [
                "INFORMATION",
                "ERROR"
              ],
              "log_group_name": "System",
              "log_stream_name": "System",
              "event_format": "xml"
            },
            {
              "event_name": "CustomizedName",
              "event_levels": [
                "WARNING",
                "ERROR"
              ],
              "log_group_name": "CustomizedLogGroup",
              "log_stream_name": "CustomizedLogStream",
              "event_format": "xml"
            }
          ]
        }
      },
      "log_stream_name": "example_log_stream_name"
    }
  }
```

手动保存 CloudWatch 代理配置文件

如果手动创建或编辑 CloudWatch 代理配置文件，则可以给它指定任何名称。为了简化故障排除，我们建议您在 Linux 服务器上将其命名为 `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`，在运行 Windows Server 的服务器上将其命名为 `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json`。创建该文件后，您可以将其复制到要运行代理的其他服务器上。

将 CloudWatch 代理配置文件上传到 Systems Manager Parameter Store

如果您计划使用 SSM Agent 在服务器上安装 CloudWatch 代理，则在手动编辑 CloudWatch 代理配置文件后，可以将其上传到 Systems Manager Parameter Store。为此，您可以使用 Systems Manager `put-parameter` 命令。

要能够将该文件存储在 Parameter Store 中，您必须使用具有足够权限的 IAM 角色。有关更多信息，请参阅 创建 IAM 角色和用户以用于 CloudWatch 代理。

可以使用以下命令，其中 *`parameter name`* 是在 Parameter Store 中用于该文件的名称，而 *`configuration_file_pathname`* 是已编辑的配置文件的路径和文件名。

```
aws ssm put-parameter --name "parameter name" --type "String" --value
 file://configuration_file_pathname
```

启用 CloudWatch Application Signals

使用 CloudWatch Application Signals 自动检测 AWS 上的应用程序，以便根据业务目标跟踪应用程序性能。Application Signals 为您提供统一的、以应用程序为中心的视图，包括您的 Java 应用程序、其依赖项和边缘。有关更多信息，请参阅 Application Signals。

CloudWatch Application Signals 利用 CloudWatch 代理从自动检测的应用程序接收指标和跟踪，您还可以选择应用规则来减少高基数数据，然后将处理后的遥测数据发布到 CloudWatch。您可以使用代理配置文件为 CloudWatch 代理提供专门针对 Application Signals 的自定义配置。首先，如果代理配置文件 `logs` 部分中的 `metrics_collected` 部分下存在 `application_signals` 部分，则表明 CloudWatch 代理将从自动检测的应用程序接收指标。同样，如果代理配置文件 `traces` 部分中的 `traces_collected` 部分下存在 `application_signals` 部分，则表明已启用 CloudWatch 代理来接收来自自动检测的应用程序的跟踪。此外，您可以选择传入自定义配置规则，以减少发布本节所述的高基数遥测数据。

- 对于 Amazon EKS 集群，当您安装 Amazon CloudWatch Observability EKS 附加组件时，CloudWatch 代理将默认启用，以接收来自自动检测的应用程序的指标和跟踪。如果您想选择传

入自定义配置规则，则可以在创建自定义代理配置时将，自定义代理配置传递给 Amazon EKS 附加组件，或者使用其他配置对其进行更新，如 （可选）其他配置 中所述。

- 对于其他支持的平台（包括 Amazon EC2），您必须使用代理配置启动 CloudWatch 代理，该代理配置可通过指定 `application_signals` 部分和任何自定义配置规则（可选）来启用 Application Signals，如本节后文所述。

以下是 CloudWatch 代理配置文件中与 CloudWatch Application Signals 相关的字段的概述。

- `logs`
  - `metrics_collected`：此字段可包含特定部分，用于指定代理将收集日志以启用使用案例，例如 CloudWatch Application Signals 和针对 Amazon EKS 增强了可观测性的 Container Insights。

> **Note**
>
> 以前，此部分还用于指定代理将收集采用嵌入式指标格式的日志。不再需要这些设置。

    - `application_signals`（可选）：指定您要启用 CloudWatch Application Signals 以接收来自自动检测的应用程序的指标，以促进 CloudWatch Application Signals。
      - `rules`（可选）：一组规则，用于有条件地选择指标和跟踪，并应用操作来应对高基数场景。每个规则都可以包含以下字段：
        - `rule_name`（可选）：规则的名称。
        - `selectors`（可选）：一组指标和跟踪维度匹配程序。每个选择器必须提供以下字段：
          - `dimension`（如果 `selectors` 不为空，则为必填项）：这指定了用作筛选条件的指标和跟踪的维度。
          - `match`（如果 `selectors` 不为空，则为必填项）：用于匹配指定维度的值的通配符模式。
        - `action`（可选）：要应用于与指定选择器匹配的指标和跟踪的操作。`action` 的值必须是以下关键字之一：
          - `keep` 指定仅将 `selectors` 匹配的指标和跟踪发送到 CloudWatch。
          - `drop` 指定删除与 `selectors` 匹配的指标和跟踪。
          - `replace` 指定替换与 `selectors` 匹配的指标和跟踪的维度。这些维度将根据 `replacements` 部分进行替换。

- replacements如果 action 为 replace，则是必需的。一组维度和值对，当 action 为 replace 时，将应用于与指定 selectors 匹配的指标和跟踪。每个替换必须提供以下字段：
  - target_dimension（如果 replacements 不为空，则为必填项）：指定需要替换的维度。
  - value（如果 replacements 不为空，则为必填项）：用于替换 target_dimension 原始值的值。
- limiter（可选）使用此部分可限制 Application Signals 发送到 CloudWatch 的指标和维度数量，以优化您的成本。
  - disabled（可选）如果为 true，则禁用指标限制功能。默认值为 false
  - drop_threshold（可选）一个 CloudWatch 代理在一个轮换时间间隔内可以导出的每项服务的最大不同指标数量。默认值为 500。
  - rotation_interval（可选）限制器重置指标记录以进行区分计数的间隔。此项表示为带有数字序列和单位后缀的字符串。支持分数。支持的单位后缀为 s、m、h、ms、us 和 ns

    1h 的默认值为 1 小时。
  - log_dropped_metrics（可选）指定删除 Application Signals 指标时代理是否应将日志写入 CloudWatch 代理日志。默认为 false。

    > **Note**
    >
    > 要激活此日志记录，还必须将 agent 部分中的 debug 参数设置为 true。

- traces
  - traces_collected
    - application_signals 可选：指定此项可让 CloudWatch 代理接收来自您的自动检测应用程序的跟踪，从而促进 CloudWatch Application Signals。

> **Note**
>
> 尽管自定义 application_signals 规则是在 logs 部分中包含的 metrics_collected 部分下指定的，但也隐含地应用于 traces_collected 部分。同一组规则将应用于指标和跟踪。

当有多个具有不同操作的规则时，按以下顺序应用：keep，然后 drop，然后 replace。

下面是一个应用自定义规则的完整 CloudWatch 代理配置文件示例。

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "rules": [
          {
            "rule_name": "keep01",
            "selectors": [
              {
                "dimension": "Service",
                "match": "pet-clinic-frontend"
              },
              {
                "dimension": "RemoteService",
                "match": "customers-service"
              }
            ],
            "action": "keep"
          },
          {
            "rule_name": "drop01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "GET /api/customer/owners/*"
              }
            ],
            "action": "drop"
          },
          {
            "rule_name": "replace01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "PUT /api/customer/owners/*/pets/*"
              },
              {
                "dimension": "RemoteOperation",
                "match": "PUT /owners"
              }
```

```
            ],
            "replacements": [
              {
                "target_dimension": "Operation",
                "value": "PUT /api/customer/owners/{ownerId}/pets{petId}"
              }
            ],
            "action": "replace"
          }
        ]
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}
```

对于前面的示例配置文件，按如下方式处理 rules：

1. 规则 keep01 可确保将维度 Service 保留为 pet-clinic-frontend、维度 RemoteService 保留为 customers-service 的所有指标和跟踪。
2. 对于在应用 keep01 后处理的指标和跟踪，drop01 规则可确保删除维度 Operation 为 GET /api/customer/owners/* 的指标和跟踪。
3. 对于在应用 drop01 后处理的指标和跟踪，replace01 规则可更新维度 Operation 为 PUT /owners、维度 RemoteOperation 为 PUT /api/customer/owners/*/pets/* 的指标和跟踪，其 Operation 维度现在被替换为 PUT /api/customer/owners/{ownerId}/pets{petId}。

以下是 CloudWatch 配置文件的完整示例，该文件通过将指标限制更改为 100、启用已删除指标的日志记录，以及将轮换间隔设置为两小时来管理 Application Signals 中的基数。

```
{
    "logs": {
        "metrics_collected": {
            "application_signals": {
                "limiter": {
                    "disabled": false,
```

```
                    "drop_threshold": 100,
                    "rotation_interval": "2h",
                    "log_dropped_metrics": true
                }
            }
        },
        "traces": {
            "traces_collected": {
                "application_signals": {}
            }
        }
    }
}
```

收集网络性能指标

在 Linux 上运行的 EC2 实例使用 Elastic Network Adapter (ENA) 发布网络性能指标。版本 1.246396.0 和更高版本的 CloudWatch 代理使您能够将这些网络性能指标导入到 CloudWatch 中。当您将这些网络性能指标导入到 CloudWatch 时，它们将作为 CloudWatch 自定义指标收费。

有关 ENA 驱动程序的更多信息，请参阅在 Linux 实例上启用 Elastic Network Adapter (ENA) 增强联网和在 Windows 实例上启用 Elastic Network Adapter (ENA) 增强联网。

在 Linux 服务器和 Windows 服务器上，设置网络性能指标集合的方式不同。

下表列出了 ENA 适配器启用的这些网络性能指标。当 CloudWatch 代理从 Linux 实例将这些指标导入 CloudWatch 时，它会在每个指标名称的开头前置 `ethtool_`。

指标	描述
Linux 服务器上的名称：**`bw_in_allowance_exceeded`** Windows 服务器上的名称：**`Aggregate inbound BW allowance exceeded`**	因入站聚合带宽超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 `metrics_collected` 部分的 `ethtool` 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无

指标	描述
Linux 服务器上的名称：**bw_out_allowance_exceeded** Windows 服务器上的名称：**Aggregate outbound BW allowance exceeded**	因出站聚合带宽超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 `metrics_collected` 部分的 `ethtool` 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无
Linux 服务器上的名称：**conntrack_allowance_available** Windows 服务器上的名称：**Available connection tracking allowance**	报告在达到该实例类型的跟踪连接限额之前，实例可以建立的跟踪连接数。此指标仅支持使用适用于弹性网络适配器（ENA）的 Linux 驱动程序版本 2.8.1 及更高版本，且基于 Nitro 的 EC2 实例，以及使用适用于弹性网络适配器（ENA）的 Windows 驱动程序版本 2.6.0 及更高版本的计算机。 仅当您将此指标列在 CloudWatch 代理配置文件 `metrics_collected` 部分的 `ethtool` 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无

指标	描述
Linux 服务器上的名称：**ena_srd_mode** Windows 服务器上的名称：**ena srd mode**	描述启用了哪些 ENA Express 功能。有关 ENA Express 的更多信息，请参阅在 Linux 实例上使用 ENA Express 提高网络性能。值如下所示： • 0 = ENA Express 关闭，UDP 关闭 • 1 = ENA Express 开启，UDP 关闭 • 2 = ENA Express 关闭，UDP 开启 **Note** 仅在最初启用 ENA Express 并将 UDP 配置为使用该功能时，才会发生这种情况。UDP 流量将保留先前值。 • 3 = ENA Express 开启，UDP 开启
Linux 服务器上的名称：**ena_srd_eligible_tx_pkts** Windows 服务器上的名称：**ena srd eligible tx pkts**	在指定时间段内发送的符合 AWS 可扩展的可靠数据报（SRD）要求的网络数据包的数量，如下所示： • 同时支持发送和接收实例类型。 • 发送和接收实例都必须配置 ENA Express。 • 发送和接收实例必须位于同一子网上。 • 实例之间的网络路径不得包含中间件。ENA Express 目前不支持中间件。
Linux 服务器上的名称：**ena_srd_tx_pkts** Windows 服务器上的名称：**ena srd tx pkts**	指定时间段内传输的 SRD 数据包数量。
Linux 服务器上的名称：**ena_srd_rx_pkts** Windows 服务器上的名称：**ena srd rx pkts**	指定时间段内接收的 SRD 数据包数量。

指标	描述
Linux 服务器上的名称：**ena_srd_resource_utilization** Windows 服务器上的名称：**ena srd resource utilization**	实例已消耗的 SRD 并发连接的最大允许内存利用率的百分比。
Linux 服务器上的名称：**linklocal_allowance_exceeded** Windows 服务器上的名称：**Link local packet rate allowance exceeded**	由于到本地代理服务的流量的 PPS 超出网络接口的最大值而丢弃的数据包数量。这会影响流向 DNS 服务、实例元数据服务和 Amazon Time Sync Service 的流量。 仅当您将此指标列在 CloudWatch 代理配置文件 `metrics_collected` 部分的 `ethtool` 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无
Linux 服务器上的名称：**pps_allowance_exceeded** Windows 服务器上的名称：**PPS allowance exceeded**	因双向 PPS 超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 `metrics_collected` 部分的 `ethtool` 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无

Linux 设置

在 Linux 服务器上，ethtool 插件使您能够将网络性能指标导入到 CloudWatch 中。

ethtool 是一个标准的 Linux 实用工具，可以收集有关 Linux 服务器上以太网设备的统计数据。它收集的统计数据取决于网络设备和驱动程序。这些统计数据的示例包括 `tx_cnt`、`rx_bytes`、`tx_errors` 和 `align_errors`。当您将 ethtool 插件与 CloudWatch 代理一起使用时，您还可以将这些统计数据以及本部分前面列出的 EC2 网络性能指标导入到 CloudWatch 中。

> **Tip**
>
> 要查找我们的操作系统和网络设备上可用的统计数据，请使用 `ethtool -S` 命令。

当 CloudWatch 代理将指标导入 CloudWatch 时，它会为所有导入指标的名称添加一个 `ethtool_` 前缀。所以标准的 ethtool 统计数据 `rx_bytes` 在 CloudWatch 中被称为 `ethtool_rx_bytes`，EC2 网络性能指标 `bw_in_allowance_exceeded` 在 CloudWatch 中被称为 `ethtool_bw_in_allowance_exceeded`。

在 Linux 服务器上，如需导入 ethtool 指标，请为 CloudWatch 代理配置文件 `metrics_collected` 部分添加 `ethtool` 部分。`ethtool` 部分包含以下子部分：

- interface_include – 包括此部分会导致代理仅从本节中列出名称的接口收集指标。如果省略此部分，则会从 `interface_exclude` 中未列出的所有以太网接口收集指标。

  默认以太网接口为 `eth0`。
- interface_exclude – 如果包含此部分，请列出您不希望从中收集指标的以太网接口。

  ethtool 插件始终忽略环回接口。
- metrics_include – 此部分列出了要导入到 CloudWatch 中的指标。它可以包括 ethtool 收集的标准统计数据和 Amazon EC2 高精度网络指标。

以下示例显示 CloudWatch 代理配置文件的一部分。此配置收集标准的 ethtool 指标 `rx_packets` 和 `tx_packets`，以及仅来自 `eth1` 接口的 Amazon EC2 网络性能指标。

有关 CloudWatch 代理配置文件的更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

```
"metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "ethtool": {
        "interface_include": [
          "eth1"
        ],
        "metrics_include": [
          "rx_packets",
```

```
            "tx_packets",
            "bw_in_allowance_exceeded",
            "bw_out_allowance_exceeded",
            "conntrack_allowance_exceeded",
            "linklocal_allowance_exceeded",
            "pps_allowance_exceeded"
          ]
       }
   }
}
```

Windows 设置

在 Windows 服务器上，网络性能指标可通过 Windows 性能计数器获得，CloudWatch 代理已经从中收集指标。因此，您无需插件即可从 Windows 服务器收集这些指标。

以下是从 Windows 收集网络性能指标的配置文件示例。有关编辑 CloudWatch 代理配置文件的更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

```
{
    "metrics": {
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "ENA Packets Shaping": {
                "measurement": [
                    "Aggregate inbound BW allowance exceeded",
                    "Aggregate outbound BW allowance exceeded",
                    "Connection tracking allowance exceeded",
                    "Link local packet rate allowance exceeded",
                    "PPS allowance exceeded"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            }
        }
    }
}
```

查看网络性能指标

将网络性能指标导入到 CloudWatch 后，您可以以时间序列图形的形式查看这些指标，并创建可监视这些指标的告警，并在它们超出指定的阈值时通知您。以下程序介绍了如何以时间序列图形的方式查看 ethtool 指标。有关设置告警的更多信息，请参阅 使用 Amazon CloudWatch 告警。

由于所有这些指标都是聚合计数器，因此您可以使用 CloudWatch 指标数学函数（如 `RATE(METRICS())`）在图形中计算这些指标的比率，或使用它们设置告警。有关指标数学函数的更多信息，请参阅 使用指标数学。

在 CloudWatch 控制台中查看网络性能指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 为该代理收集的指标选择命名空间。默认情况下，该命名空间为 CWAgent，但您可能已经在 CloudWatch 代理配置文件中指定了不同的命名空间。
4. 选择指标维度（例如 Per-Instance Metrics（每个实例的指标））。
5. All metrics 选项卡显示命名空间中该维度的所有指标。您可执行以下操作：
   a. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。
   b. 要对表进行排序，请使用列标题。
   c. 要按资源进行筛选，请选择资源 ID，然后选择添加到搜索。
   d. 要按指标进行筛选，请选择指标名称，然后选择添加到搜索。
6. （可选）要将此图表添加到 CloudWatch 控制面板，请选择 Actions（操作），然后选择 Add to dashboard（添加到控制面板）。

收集 NVIDIA GPU 指标

您可以使用 CloudWatch 代理从 Linux 服务器收集 NVIDIA GPU 指标。要对此进行设置，请在 CloudWatch 代理配置文件的 `metrics_collected` 部分中添加 `nvidia_gpu` 部分。有关更多信息，请参阅 Linux 部分。

此外，该实例必须已经安装 NVIDIA 驱动程序。某些亚马逊机器映像（AMI）上已经预装 NVIDIA 驱动程序。如果没有安装，您可以手动安装该驱动程序。有关更多信息，请参见在 Linux 实例上安装 NVIDIA 驱动程序。

可以收集以下指标。所有这些指标都在没有 CloudWatch Unit 的情况下收集，但是您可以通过向 CloudWatch 代理配置文件添加参数来为每个指标指定单位。有关更多信息，请参阅 Linux 部分。

指标	CloudWatch 的指标名称	描述
utilization_gpu	nvidia_smi_utilization_gpu	在过去的采样周期内 GPU 上的一个或多个内核运行的时间百分比。
temperature_gpu	nvidia_smi_temperature_gpu	GPU 核心温度（以摄氏度为单位）。
power_draw	nvidia_smi_power_draw	上次测量的整个显卡功耗（以瓦为单位）。
utilization_memory	nvidia_smi_utilization_memory	在过去的样本周期内读取或写入全局（设备）内存的时间百分比。
fan_speed	nvidia_smi_fan_speed	设备风扇目前预计以最大风扇速度运行的百分比。
memory_total	nvidia_smi_memory_total	报告的总内存（以 MB 为单位）。
memory_used	nvidia_smi_memory_used	已使用的内存（以 MB 为单位）。
memory_free	nvidia_smi_memory_free	空闲内存（以 MB 为单位）。
pcie_link_gen_current	nvidia_smi_pcie_link_gen_current	当前链接生成。
pcie_link_width_current	nvidia_smi_pcie_link_width_current	当前链接宽度。

指标	CloudWatch 的指标名称	描述
encoder_stats_session_count	nvidia_smi_encoder_stats_session_count	当前编码器会话数量。
encoder_stats_average_fps	nvidia_smi_encoder_stats_average_fps	每秒编码帧数的移动平均值。
encoder_stats_average_latency	nvidia_smi_encoder_stats_average_latency	编码延迟的移动平均值（以微秒为单位）。
clocks_current_graphics	nvidia_smi_clocks_current_graphics	显卡（着色器）时钟的当前频率。
clocks_current_sm	nvidia_smi_clocks_current_sm	流式多处理器（SM）时钟的当前频率。
clocks_current_memory	nvidia_smi_clocks_current_memory	内存时钟的当前频率。
clocks_current_video	nvidia_smi_clocks_current_video	视频（编码器加解码器）时钟的当前频率。

所有这些指标都使用以下维度收集:

维度	描述
index	此服务器上 GPU 的唯一标识符。表示设

维度	描述
	备的 NVIDIA 管理库（NVML）索引。
name	GPU 类型。例如，NVIDIA Tesla A100
host	服务器主机名。

使用 procstat 插件收集流程指标

procstat 插件让您能够从各个流程收集指标。这在 Linux 服务器上以及运行支持的 Windows Server 版本的服务器上受支持。

主题

- 配置 CloudWatch 代理用于 procstat
- Procstat 收集的指标
- 查看由 CloudWatch 代理导入的进程指标

配置 CloudWatch 代理用于 procstat

要使用 procstat 插件，请在 CloudWatch 代理配置文件的 `metrics_collected` 部分中添加 `procstat` 部分。您可以通过以下三种方式指定要监控的进程。您可以仅使用其中一个方法，但您可以使用该方法来指定一个或多个要监控的进程。

- `pid_file`：按照它们创建的进程标识号 (PID) 文件的名称选择进程。
- `exe`：使用正则表达式匹配规则，选择进程名称与您指定的字符串相匹配的进程。匹配是“包含”匹配，这意味着如果您指定 `agent` 作为要匹配的术语，则具有类似于 `cloudwatchagent` 的名称的进程将匹配该术语。有关更多信息，请参阅语法。
- `pattern`：通过用于启动进程的命令行选择进程。将使用正则表达式匹配规则选择命令行与指定的字符串匹配的所有进程。将检查整个命令行，包括用于命令行的参数和选项。

匹配是“包含”匹配，这意味着如果您指定 -c 作为要匹配的术语，则具有类似于 -config 的参数的进程将匹配该术语。

- drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

CloudWatch 代理仅使用其中一个方法，即使您包含多个以上部分。如果您指定多个部分，CloudWatch 代理将使用 pid_file 部分（如果存在）。否则，它使用 exe 部分。

在 Linux 服务器上，您在 exe 或 pattern 部分中指定的字符串的计算结果为正则表达式。在运行 Windows Server 的服务器上，这些字符串的计算结果为 WMI 查询。例如，pattern: "%apache%" 就是一个示例。有关更多信息，请参阅 LIKE 运算符。

无论使用哪种方法，您都可以包含一个可选 metrics_collection_interval 参数，该参数以秒为单位指定收集这些指标的频率。如果省略该参数，则使用 60 秒的默认值。

在以下部分的示例中，procstat 部分是唯一包含在代理配置文件的 metrics_collected 部分中的部分。实际配置文件也在 metrics_collected 中包含其他部分。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

使用 pid_file 进行配置

以下示例 procstat 部分监控创建 PID 文件 example1.pid 和 example2.pid 的进程。从每个进程收集不同的指标。默认情况下，每 10 秒收集一次从创建 example2.pid 的进程中收集的指标，每 60 秒收集一次从 example1.pid 进程中收集的指标。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pid_file": "/var/run/example1.pid",
                    "measurement": [
                        "cpu_usage",
                        "memory_rss"
                    ]
                },
                {
```

```
                    "pid_file": "/var/run/example2.pid",
                    "measurement": [
                        "read_bytes",
                        "read_count",
                        "write_bytes"
                    ],
                    "metrics_collection_interval": 10
                }
            ]
        }
    }
}
```

使用 exe 进行配置

以下示例 procstat 部分监控名称与字符串 agent 或 plugin 相匹配的所有进程。从每个进程收集相同的指标。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "exe": "agent",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                },
                {
                    "exe": "plugin",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                }
            ]
        }
    }
}
```

使用模式进行配置

以下示例 procstat 部分监控命令行与字符串 config 或 -c 相匹配的所有进程。从每个进程收集相同的指标。

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pattern": "config",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                },
                {
                    "pattern": "-c",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                }
            ]
        }
    }
}
```

Procstat 收集的指标

下表列出可以使用 procstat 插件收集的指标。

CloudWatch 代理将 procstat 添加到以下指标名称的开头。有不同的语法，具体取决于是从 Linux 服务器还是从运行 Windows Server 的服务器收集指标。例如，从 Linux 中收集时，cpu_time 指标显示为 procstat_cpu_time；从 Windows Server 中收集时，它显示为 procstat cpu_time。

指标名称	可用于	描述
cpu_time	Linux	进程使用 CPU 的时间量。该指标以百分之一秒为单位。 单位：计数
cpu_time_guest	Linux	进程处于来宾模式的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_guest_nice	Linux	进程在具有 nice 值的来宾模式下运行的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_idle	Linux	进程处于空闲模式的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无

指标名称	可用于	描述
cpu_time_iowait	Linux	进程等待 I/O 操作完成的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_irq	Linux	进程处理中断的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_nice	Linux	进程处于 nice 模式的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_soft_irq	Linux	进程处理软件中断的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无

指标名称	可用于	描述
cpu_time_steal	Linux	在虚拟化环境中运行时，在其他操作系统上运行的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_stolen	Linux、Windows Server	进程的被抢占时间，这是在虚拟化环境中的其他操作系统上所花的时间。该指标以百分之一秒为单位。 类型：浮点值 单位：无
cpu_time_system	Linux、Windows Server、macOS	进程处于系统模式的时间量。该指标以百分之一秒为单位。 类型：浮点值 单位：计数

指标名称	可用于	描述
cpu_time_user	Linux、Windows Server、macOS	进程处于用户模式的时间量。该指标以百分之一秒为单位。 单位：计数
cpu_usage	Linux、Windows Server、macOS	进程在任何容量中处于活动状态的时间百分比。 单位：百分比
memory_data	Linux、macOS	进程用于数据的内存量。 单位：字节
memory_locked	Linux、macOS	进程已锁定的内存量。 单位：字节
memory_rss	Linux、Windows Server、macOS	进程使用的实际内存量（驻留集）。 单位：字节
memory_stack	Linux、macOS	进程使用的堆栈内存量。 单位：字节

指标名称	可用于	描述
memory_swap	Linux、macOS	进程使用的交换内存量。 单位：字节
memory_vms	Linux、Windows Server、macOS	进程使用的虚拟内存量。 单位：字节
num_fds	Linux	此进程打开的文件描述符数量。 单位：无
num_threads	Linux、Windows、macOS	进程中的线程数。 单位：无
pid	Linux、Windows Server、macOS	进程标识符 (ID)。 单位：无

指标名称	可用于	描述
pid_count	Linux、Windows Server、macOS	与此进程关联的进程 ID 的数量。 在 Linux 服务器和 macOS 电脑上，此指标的完整名称为 procstat_lookup_pid_count ；在 Windows 服务器上则为 procstat_lookup pid_count 。 单位：无
read_bytes	Linux、Windows Server	进程已从磁盘中读取的字节数。 单位：字节
write_bytes	Linux、Windows Server	进程已写入到磁盘的字节数。 单位：字节
read_count	Linux、Windows Server	进程已执行的磁盘读取操作的数目。 单位：无

指标名称	可用于	描述
rlimit_realtime_priority_hard	Linux	可以为此进程设置的实时优先级硬限制。 单位：无
rlimit_realtime_priority_soft	Linux	可以为此进程设置的实时优先级软限制。 单位：无
rlimit_signals_pending_hard	Linux	此进程可以排队的最大信号数硬限制。 单位：无
rlimit_signals_pending_soft	Linux	此进程可以排队的最大信号数软限制。 单位：无
rlimit_nice_priority_hard	Linux	此进程可以设置的最大 nice 优先级硬限制。 单位：无
rlimit_nice_priority_soft	Linux	此进程可以设置的最大 nice 优先级软限制。 单位：无

指标名称	可用于	描述
rlimit_num_fds_hard	Linux	此进程可以打开的最大文件描述符数硬限制。 单位：无
rlimit_num_fds_soft	Linux	此进程可以打开的最大文件描述符数软限制。 单位：无
write_count	Linux、Windows Server	进程已执行的磁盘写入操作的数目。 单位：无
involuntary_context_switches	Linux	进程不自觉上下文切换的次数。 单位：无
voluntary_context_switches	Linux	进程自觉上下文切换的次数。 单位：无
realtime_priority	Linux	进程的实时优先级的当前使用率。 单位：无

指标名称	可用于	描述
nice_priority	Linux	进程的良好状态优先级的当前使用率。 单位：无
signals_pending	Linux	等待由进程处理的信号数量。 单位：无
rlimit_cpu_time_hard	Linux	进程的硬 CPU 时间资源限制。 单位：无
rlimit_cpu_time_soft	Linux	进程的软 CPU 时间资源限制。 单位：无
rlimit_file_locks_hard	Linux	进程的硬文件锁资源限制。 单位：无
rlimit_file_locks_soft	Linux	进程的软文件锁资源限制。 单位：无

指标名称	可用于	描述
rlimit_memory_data_hard	Linux	进程上用于数据的内存的硬资源限制。 单位：字节
rlimit_memory_data_soft	Linux	进程上用于数据的内存的软资源限制。 单位：字节
rlimit_memory_locked_hard	Linux	进程上用于锁定的内存的硬资源限制。 单位：字节
rlimit_memory_locked_soft	Linux	进程上用于锁定的内存的软资源限制。 单位：字节
rlimit_memory_rss_hard	Linux	进程上用于物理内存的硬资源限制。 单位：字节
rlimit_memory_rss_soft	Linux	进程上用于物理内存的软资源限制。 单位：字节

指标名称	可用于	描述
rlimit_memory_stack_hard	Linux	进程堆栈的硬资源限制。 单位：字节
rlimit_memory_stack_soft	Linux	进程堆栈的软资源限制。 单位：字节
rlimit_memory_vms_hard	Linux	进程上用于虚拟内存的硬资源限制。 单位：字节
rlimit_memory_vms_soft	Linux	进程上用于虚拟内存的软资源限制。 单位：字节

查看由 CloudWatch 代理导入的进程指标

将进程指标导入到 CloudWatch 后，您可以以时间序列图形的形式查看这些指标，并创建可监视这些指标的告警，并在它们超出指定的阈值时通知您。以下程序介绍了如何以时间序列图形的方式查看进程指标。有关设置告警的更多信息，请参阅 使用 Amazon CloudWatch 告警。

在 CloudWatch 控制台中查看进程指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 为该代理收集的指标选择命名空间。默认情况下，该命名空间为 CWAgent，但您可能已经在 CloudWatch 代理配置文件中指定了不同的命名空间。
4. 选择指标维度（例如 Per-Instance Metrics（每个实例的指标））。
5. All metrics 选项卡显示命名空间中该维度的所有指标。您可执行以下操作：

a. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。

b. 要对表进行排序，请使用列标题。

c. 要按资源进行筛选，请选择资源 ID，然后选择 Add to search。

d. 要按指标进行筛选，请选择指标名称，然后选择 Add to search。

6. （可选）要将此图表添加到 CloudWatch 控制面板，请选择 Actions（操作），然后选择 Add to dashboard（添加到控制面板）。

使用 StatsD 检索自定义指标

您可以将 CloudWatch 代理和 StatsD 协议结合使用，以从应用程序或服务中检索附加的自定义指标。StatSD 是一种流行的开源解决方案，可以从各种应用程序中收集指标。StatSD 对于检测您自己的指标特别有用。有关将 CloudWatch 代理和 StatSD 结合使用的示例，请参阅如何使用 Amazon CloudWatch 代理更好地监控您的自定义应用程序指标。

StatsD 在 Linux 服务器以及运行 Windows Server 的服务器上受支持。CloudWatch 支持以下 StatsD 格式：

```
MetricName:value|type|@sample_rate|#tag1:
  value,tag1...
```

- MetricName – 一个没有冒号、竖线、# 字符或 @ 字符的字符串。
- value – 它可以是整数或浮点数。
- type – 指定 c 表示计数器，g 表示计量器，ms 表示计时器，h 表示直方图，s 表示集合。
- sample_rate –（可选）一个介于 0 和 1 之间的浮点数（含）。仅用于计数器、直方图和计时器指标。默认值为 1（采样时间为 100%）。
- tags –（可选）一个逗号分隔的标签列表。StatsD 标签类似于 CloudWatch 中的维度。对键/值标签使用冒号，例如 env:prod。

您可以使用任何遵循此格式的 StatsD 客户端将指标发送到 CloudWatch 代理。有关一些可用 StatsD 客户端的更多信息，请参阅 GitHub 上的 StatsD 客户端页面。

要收集这些自定义指标，请将 "statsd": {} 行添加到代理配置文件的 metrics_collected 部分。您可以手动添加此行。如果您使用向导创建配置文件，向导会为您完成此操作。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。

StatsD 默认配置适用于大多数用户。有一些可选字段，您可以根据需要将其添加到代理配置文件的 statsd 部分：

- service_address – CloudWatch 代理应该侦听的服务地址。格式为 *ip*:*port*。如果您忽略了 IP 地址，则该代理将侦听所有可用接口。只有 UDP 格式受支持，因此您不需要指定 UDP 前缀。

  默认值为 :8125。
- metrics_collection_interval – StatsD 插件运行和收集指标的频率（秒）。默认值为 10 秒。范围为 1 – 172000。
- metrics_aggregation_interval – CloudWatch 将指标聚合为单个数据点的频率（秒）。默认值为 60 秒。

  例如，如果 metrics_collection_interval 为 10，并且 metrics_aggregation_interval 为 60，则 CloudWatch 每 10 秒收集数据。在每分钟之后，来自该分钟的六个数据读数被聚合到单个数据点中，该数据点被发送到 CloudWatch。

  范围为 1 – 172000。将 metrics_aggregation_interval 设置为 0 会禁用 StatsD 指标的聚合。
- allowed_pending_messages – 允许排队的 UDP 消息数量。当队列已满时，StatsD 服务器会开始丢弃数据包。默认值是 10000。
- drop_original_metrics – 可选。如果您使用 metrics 部分中的 aggregation_dimensions 字段将指标以汇总为聚合结果，则默认情况下，代理会同时发送聚合指标和原始指标，原始指标是按维度的每个值分开的。如果您不希望将原始指标发送到 CloudWatch，则可以指定此参数和指标列表。同时指定了此参数的指标将不会将其各维度的指标报告给 CloudWatch，而只报告聚合指标。这可以减少代理要采集的指标数量，从而降低成本。

下面是代理配置文件的 statsd 部分的示例，使用默认端口、自定义集合和聚合间隔。

```
{
   "metrics":{
      "metrics_collected":{
         "statsd":{
            "service_address":":8125",
            "metrics_collection_interval":60,
            "metrics_aggregation_interval":300
         }
      }
   }
```

```
}
```

查看由 CloudWatch 代理导入的 StatsD 指标

将 StatsD 指标导入到 CloudWatch 后，您可以以时间序列图形的形式查看这些指标，并创建可监视这些指标的告警，并在它们超出指定的阈值时通知您。以下程序介绍了如何以时间序列图形的方式查看 StatsD 指标。有关设置告警的更多信息，请参阅 使用 Amazon CloudWatch 告警。

在 CloudWatch 控制台中查看 StatsD 指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 为该代理收集的指标选择命名空间。默认情况下，该命名空间为 CWAgent，但您可能已经在 CloudWatch 代理配置文件中指定了不同的命名空间。
4. 选择指标维度（例如 Per-Instance Metrics（每个实例的指标））。
5. All metrics 选项卡显示命名空间中该维度的所有指标。您可执行以下操作：
   a. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。
   b. 要对表进行排序，请使用列标题。
   c. 要按资源进行筛选，请选择资源 ID，然后选择 Add to search。
   d. 要按指标进行筛选，请选择指标名称，然后选择 Add to search。
6. （可选）要将此图表添加到 CloudWatch 控制面板，请选择 Actions（操作），然后选择 Add to dashboard（添加到控制面板）。

使用 collectd 检索自定义指标

您可以使用 CloudWatch 代理以及 collectd 协议（该协议仅在 Linux 服务器上受支持）从应用程序或服务中检索其他指标。collectd 是一种通用的开源解决方案，其中的插件可以收集各种应用程序的系统统计数据。通过将 CloudWatch 代理可以收集的系统指标与 collectd 中的其他指标结合起来，您可以更好地监控、分析系统和应用程序并进行故障排除。有关 collectd 的更多信息，请参阅 collectd - 系统统计数据收集守护程序。

您可以使用 collectd 软件将指标发送到 CloudWatch 代理。对于 collectd 指标，CloudWatch 代理充当服务器，而 collectd 插件充当客户端。

collectd 软件不会自动安装在每个服务器上。在运行 Amazon Linux 2 的服务器上，请按照下列步骤安装 collectd

```
sudo amazon-linux-extras install collectd
```

有关在其他系统上安装 collectd 的信息，请参阅 collectd 下载页面。

要收集这些自定义指标，请将 "collectd": {} 行添加到代理配置文件的 metrics_collected 部分。您可以手动添加此行。如果您使用向导创建配置文件，向导会为您完成此操作。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。

还将提供可选参数。如果您使用的是 collectd 并且未将 `/etc/collectd/auth_file` 用作 collectd_auth_file，则必须设置其中一些选项。

- service_address：CloudWatch 代理应该侦听的服务地址。格式为 "`udp://`*`ip`*`:`*`port`*。默认为 `udp://127.0.0.1:25826`。
- name_prefix：附加到每个 collectd 指标名称开头的前缀。默认为 `collectd_`。最大长度为 255 个字符。
- collectd_security_level：设置网络通信的安全级别。默认值为 encrypt (加密)。

  encrypt (加密) 指定只接受加密的数据。sign (签名) 指定只接受已签名且加密的数据。none (无) 指定接受所有数据。如果为 collectd_auth_file 指定值，则会尽可能解密加密数据。

  有关更多信息，请参阅 collectd Wiki 中的客户端设置和可能的交互。
- collectd_auth_file 设置用户名映射到密码的文件。这些密码用于验证签名和解密已加密的网络数据包。如果给定，则会验证签名的数据并解密加密的数据包。否则，会接受签名的数据而不检查签名，并且加密的数据不能被解密。

  默认为 `/etc/collectd/auth_file`。

  如果 collectd_security_level 设置为 none (无)，则这是可选的。如果将 collectd_security_level 设置为 `encrypt` 或 sign (签名)，则必须指定 collectd_auth_file。

  对于身份验证文件的格式，每行是一个用户名，后跟一个冒号和任意数量的空格，再后跟密码。例如：

  ```
  user1: user1_password

  user2: user2_password
  ```

- collectd_typesdb：包含数据集描述的一个或多个文件的列表。即使列表中只有一个条目，列表也必须用括号括起来。列表中的每个条目都必须用双引号括起来。如果有多个条目，请用逗号分隔。Linux 服务器上的默认值为 `["/usr/share/collectd/types.db"]`。macOS 电脑上的默认值取决于 collectd 的版本。例如，`["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]`。

  有关更多信息，请参阅 https://www.collectd.org/documentation/manpages/types.db.html。
- metrics_aggregation_interval：CloudWatch 将指标聚合为单个数据点的频率（秒）。默认值为 60 秒。范围为 0 至 172,000。将它设置为 0 会禁用 collectd 指标的聚合。

下面是代理配置文件的 collectd 部分的示例。

```
{
   "metrics":{
      "metrics_collected":{
         "collectd":{
            "name_prefix":"My_collectd_metrics_",
            "metrics_aggregation_interval":120
         }
      }
   }
}
```

查看由 CloudWatch 代理导入的已收集指标

将 collected 指标导入到 CloudWatch 后，您可以以时间序列图形的形式查看这些指标，并创建可监视这些指标的告警，并在它们超出指定的阈值时通知您。以下程序介绍了如何以时间序列图形的方式查看 collected 指标。有关设置告警的更多信息，请参阅 使用 Amazon CloudWatch 告警。

在 CloudWatch 控制台中查看 collected 指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 为该代理收集的指标选择命名空间。默认情况下，该命名空间为 CWAgent，但您可能已经在 CloudWatch 代理配置文件中指定了不同的命名空间。
4. 选择指标维度（例如 Per-Instance Metrics（每个实例的指标））。
5. All metrics 选项卡显示命名空间中该维度的所有指标。您可执行以下操作：

a. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。

b. 要对表进行排序，请使用列标题。

c. 要按资源进行筛选，请选择资源 ID，然后选择 Add to search。

d. 要按指标进行筛选，请选择指标名称，然后选择 Add to search。

6. （可选）要将此图表添加到 CloudWatch 控制面板，请选择 Actions（操作），然后选择 Add to dashboard（添加到控制面板）。

在 Amazon EC2 实例上设置和配置 Prometheus 指标集合

以下部分介绍如何在 EC2 实例上安装具有 Prometheus 监控功能的 CloudWatch 代理，以及如何配置代理以抓取其他目标。它还提供了设置示例工作负载以使用 Prometheus 监控进行测试的教程。

有关 CloudWatch 代理支持的操作系统的信息，请参阅 使用 CloudWatch 代理收集指标、日志和跟踪信息

VPC 安全组要求

如果您使用的是 VPC，则以下要求适用。

- Prometheus 工作负载的安全组的入口规则必须向 CloudWatch 代理打开 Prometheus 端口，以便通过私有 IP 抓取 Prometheus 指标。
- CloudWatch 代理的安全组的出口规则必须允许 CloudWatch 代理通过私有 IP 连接到 Prometheus 工作负载的端口。

主题

- 步骤 1：安装 CloudWatch 代理
- 步骤 2：抓取 Prometheus 源并导入指标
- 示例：为 Prometheus 指标测试设置 Java/JMX 示例工作负载

步骤 1：安装 CloudWatch 代理

第一步是在 EC2 实例上安装 CloudWatch 代理。有关说明，请参阅 安装 CloudWatch 代理。

步骤 2：抓取 Prometheus 源并导入指标

具有 Prometheus 监控功能的 CloudWatch 代理需要两种配置来抓取 Prometheus 指标。一种是 Prometheus 文档的 <scrape_config> 中记录的标准 Prometheus 配置。另一种配置是 CloudWatch 代理配置文件。

Prometheus 抓取配置

CloudWatch 代理支持标准的 Prometheus 抓取配置，如 Prometheus 文档中的 <scrape_config> 所述。您可以编辑此部分以更新此文件中已有的配置，并添加其他 Prometheus 抓取目标。示例配置文件包含以下全局配置行：

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
    - files: ["C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_1.yaml",
 "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_2.yaml"]
```

`global` 部分指定在所有配置上下文中有效的参数。它们还用作其他配置部分的默认值。它包含以下参数：

- `scrape_interval` – 定义抓取目标的频率。
- `scrape_timeout` – 定义在抓取请求超时之前等待的时间。

`scrape_configs` 部分指定了一组目标和参数，用于定义如何抓取它们。它包含以下参数：

- `job_name` – 默认情况下分配给已抓取指标的任务名称。
- `sample_limit` – 将被接受的抓取样本数量的每次抓取限制。
- `file_sd_configs` – 文件服务发现配置的列表。它读取一组包含零个或多个静态配置列表的文件。`file_sd_configs` 部分包含 `files` 参数，该参数定义从中提取目标组的文件的模式。

CloudWatch 代理支持以下服务发现配置类型。

`static_config` 允许指定目标列表及其公用标签集。它是在抓取配置中指定静态目标的规范方法。

以下是用于从本地主机中抓取 Prometheus 指标的示例静态配置。如果 Prometheus 端口对运行代理的服务器打开，也可以从其他服务器中抓取指标。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml
- targets:
    - 127.0.0.1:9404
  labels:
    key1: value1
    key2: value2
```

该示例包括以下参数：

- `targets` – 由静态配置抓取的目标。
- `labels` – 分配给从目标中抓取的所有指标的标签。

`ec2_sd_config` 允许从 Amazon EC2 实例中检索抓取目标。以下为示例 `ec2_sd_config` 从 EC2 实例列表中抓取 Prometheus 指标。这些实例的 Prometheus 端口必须向运行 CloudWatch 代理的服务器打开。运行 CloudWatch 代理的 EC2 实例的 IAM 角色必须包含 `ec2:DescribeInstance` 权限。例如，您可以将托管式策略 AmazonEC2ReadOnlyAccess 附加到运行 CloudWatch 代理的实例。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

该示例包括以下参数：

- `region` – 目标 EC2 实例所在的 AWS 区域。如果将此项保留空白，则使用实例元数据中的区域。
- `port` – 要从中抓取指标的端口。

- `filters` – 用于筛选实例列表的可选筛选条件。此示例基于 EC2 实例 ID 进行筛选。有关可以筛选的更多条件，请参阅 DescribeInstances。

Prometheus 的 CloudWatch 代理配置

CloudWatch 代理配置文件包含 `logs` 和 `metrics_collected` 下的 `prometheus` 部分。它包括以下参数。

- cluster_name – 指定要在日志事件中添加为标签的集群名称。该字段是可选的。
- log_group_name – 指定已抓取 Prometheus 指标的日志组名称。
- prometheus_config_path – 指定 Prometheus 抓取配置文件路径。
- emf_processor – 指定嵌入式指标格式处理器配置。有关嵌入式指标格式的更多信息，请参阅在日志中嵌入指标。

  `emf_processor` 部分可能包括以下参数：

  - metric_declaration_dedup – 设置为 true，则启用嵌入式指标格式指标的重复数据消除功能。
  - metric_namespace – 指定发射的 CloudWatch 指标的指标命名空间。
  - metric_unit – 指定指标名称：指标单位映射。有关受支持指标单位的信息，请参阅 MetricDatum。
  - metric_declaration – 是指定要生成的采用嵌入式指标格式的日志数组的部分。默认情况下，CloudWatch 代理从中进行导入的每个 Prometheus 源都有 `metric_declaration` 部分。这些部分各包括以下字段：
    - `source_labels` 指定由 `label_matcher` 行检查的标签的值。
    - `label_matcher` 是一个正则表达式，用于检查 `source_labels` 中列出的标签的值。匹配的指标将启用，以包含在发送到 CloudWatch 的嵌入式指标格式中。
    - `metric_selectors` 是一个正则表达式，用于指定要收集并发送到 CloudWatch 的指标。
    - `dimensions` 是要用作每个选定指标的 CloudWatch 维度的标签列表。

以下是 Prometheus 的 CloudWatch 代理配置示例。

```
{
   "logs":{
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-cluster",
            "log_group_name":"Prometheus",
```

```
            "prometheus_config_path":"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent-Prometheus",
               "metric_unit":{
                  "jvm_threads_current": "Count",
                  "jvm_gc_collection_seconds_sum": "Milliseconds"
               },
               "metric_declaration":[
                  {
                     "source_labels":[
                        "job", "key2"
                     ],
                     "label_matcher":"MY_JOB;^value2",
                     "dimensions":[
                        [
                           "key1", "key2"
                        ],
                        [
                           "key2"
                        ]
                     ],
                     "metric_selectors":[
                        "^jvm_threads_current$",
                        "^jvm_gc_collection_seconds_sum$"
                     ]
                  }
               ]
            }
         }
      }
   }
}
```

前一个示例配置嵌入式指标格式部分，以便在满足以下条件时作为日志事件发送：

- 标签 job 的值为 MY_JOB
- 标签 key2 的值为 value2
- Prometheus 指标 jvm_threads_current 和 jvm_gc_collection_seconds_sum 同时包含 job 和 key2 标签。

发送的日志事件包括以下突出显示的部分。

```
{
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Count",
                    "Name": "jvm_threads_current"
                },
                {
                    "Unit": "Milliseconds",
                    "Name": "jvm_gc_collection_seconds_sum"
                }
            ],
            "Dimensions": [
                [
                    "key1",
                    "key2"
                ],
                [
                    "key2"
                ]
            ],
            "Namespace": "CWAgent-Prometheus"
        }
    ],
    "ClusterName": "prometheus-cluster",
    "InstanceId": "i-0e45bd06f196096c8",
    "Timestamp": "1607966368109",
    "Version": "0",
    "host": "EC2AMAZ-PDDOIUM",
    "instance": "127.0.0.1:9404",
    "jvm_threads_current": 2,
    "jvm_gc_collection_seconds_sum": 0.006000000000000002,
    "prom_metric_type": "gauge",
    ...
}
```

示例：为 Prometheus 指标测试设置 Java/JMX 示例工作负载

JMX Exporter 是 Prometheus 的官方导出程序，可以将 JMX MBeans 作为 Prometheus 指标进行抓取和公开。有关详细信息，请参阅 prometheus/jmx_exporter。

CloudWatch 代理可以从 EC2 实例上的 JMX Exporter 的 Java 虚拟机 (JVM)、Hjava 和 Tomcat (Catalina) 中收集预定义的 Prometheus 指标。

步骤 1：安装 CloudWatch 代理

第一步是在 EC2 实例上安装 CloudWatch 代理。有关说明，请参阅 安装 CloudWatch 代理。

步骤 2：启动 Java/JMX 工作负载

下一步是启动 Java/JMX 工作负载。

首先，从以下位置下载最新的 JMX Exporter jar 文件：prometheus/jmx_exporter。

为您的示例应用程序使用 jar

以下各部分中的示例命令使用 `SampleJavaApplication-1.0-SNAPSHOT.jar` 作为 jar 文件。将命令的这些部分替换为应用程序的 jar。

准备 JMX Exporter 配置

`config.yaml` 文件是 JMX Exporter 配置文件。有关更多信息，请参阅 JMX Exporter 文档中的配置。

以下是 Java 和 Tomcat 的示例配置。

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
```

```
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

使用 Prometheus 导出程序启动 Java 应用程序

启动示例应用程序。这会将 Prometheus 指标发送到端口 9404。请务必将入口点 `com.gubupt.sample.app.App` 替换为您的示例 java 应用程序的正确的信息。

在 Linux 操作系统上，输入以下命令。

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp
  ./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

在 Windows 操作系统上，输入以下命令。

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp  .
\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

验证 Prometheus 指标发射

验证是否正在发射 Prometheus 指标。

在 Linux 操作系统上，输入以下命令。

```
$ curl localhost:9404
```

在 Windows 操作系统上，输入以下命令。

```
PS C:\> curl  http://localhost:9404
```

Linux 上的输出示例：

```
StatusCode        : 200
StatusDescription : OK
Content           : # HELP jvm_classes_loaded The number of classes that are currently
 loaded in the JVM
                    # TYPE jvm_classes_loaded gauge
                    jvm_classes_loaded 2526.0
                    # HELP jvm_classes_loaded_total The total number of class...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 71908
                    Content-Type: text/plain; version=0.0.4; charset=utf-8
                    Date: Fri, 18 Dec 2020 16:38:10 GMT

                    # HELP jvm_classes_loaded The number of classes that are
 currentl...
Forms             : {}
Headers           : {[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
 charset=utf-8], [Date, Fri, 18
                    Dec 2020 16:38:10 GMT]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
```

```
RawContentLength  : 71908
```

步骤 3：配置 CloudWatch 代理以抓取 Prometheus 指标

接下来，在 CloudWatch 代理配置文件中设置 Prometheus 抓取配置。

为 Java/JMX 示例设置 Prometheus 抓取配置

1. 设置 `file_sd_config` 和 `static_config` 配置。

   在 Linux 操作系统上，输入以下命令。

   ```
   $ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
   global:
     scrape_interval: 1m
     scrape_timeout: 10s
   scrape_configs:
     - job_name: jmx
       sample_limit: 10000
       file_sd_configs:
         - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
   ```

   在 Windows 操作系统上，输入以下命令。

   ```
   PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
   global:
     scrape_interval: 1m
     scrape_timeout: 10s
   scrape_configs:
     - job_name: jmx
       sample_limit: 10000
       file_sd_configs:
         - files: [ "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
   \prometheus_file_sd.yaml" ]
   ```

2. 设置抓取目标配置。

   在 Linux 操作系统上，输入以下命令。

   ```
   $ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
   - targets:
     - 127.0.0.1:9404
   ```

```
  labels:
    application: sample_java_app
    os: linux
```

在 Windows 操作系统上，输入以下命令。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

3. 通过 ec2_sc_config 设置 Prometheus 抓取配置。用正确的 EC2 实例 ID 替代 *your-ec2-instance-id*。

   在 Linux 操作系统上，输入以下命令。

```
$ cat .\prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

   在 Windows 操作系统上，输入以下命令。

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

4. 设置 CloudWatch 代理配置。首先，导航到正确的目录。在 Linux 上，它是 /opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json。在 Windows 上，它是 C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json。

   以下是定义了 Java/JHX Prometheus 指标的示例配置。请务必使用正确的路径替换 *path-to-Prometheus-Scrape-Configuration-file*。

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "prometheus": {
        "cluster_name": "my-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",
        "emf_processor": {
          "metric_declaration_dedup": true,
          "metric_namespace": "PrometheusTest",
          "metric_unit":{
            "jvm_threads_current": "Count",
            "jvm_classes_loaded": "Count",
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",
            "catalina_manager_activesessions": "Count",
            "jvm_gc_collection_seconds_sum": "Seconds",
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",
            "jvm_memory_bytes_used": "Bytes",
            "jvm_memory_pool_bytes_used": "Bytes"
          },
          "metric_declaration": [
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["instance"]],
              "metric_selectors": [
                "^jvm_threads_current$",
                "^jvm_classes_loaded$",
                "^java_lang_operatingsystem_freephysicalmemorysize$",
                "^catalina_manager_activesessions$",
                "^jvm_gc_collection_seconds_sum$",
                "^catalina_globalrequestprocessor_bytesreceived$"
              ]
```

```
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["area"]],
              "metric_selectors": [
                "^jvm_memory_bytes_used$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["pool"]],
              "metric_selectors": [
                "^jvm_memory_pool_bytes_used$"
              ]
            }
          ]
        }
      }
    },
    "force_flush_interval": 5
  }
}
```

5. 通过输入以下命令之一，重新启动 CloudWatch 代理。

   在 Linux 操作系统上，输入以下命令。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
   config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
   ```

   在 Windows 操作系统上，输入以下命令。

   ```
   & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
    -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent
   \cwagent-config.json
   ```

查看 Prometheus 指标和日志

您现在可以查看正在收集的 Java/JMX 指标。

查看 Java/JMX 示例工作负载的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在运行集群的区域中，选择左侧导航窗格中的 Metrics（指标）。查找 PrometheusTest 命名空间以查看指标。
3. 要查看 CloudWatch Logs 事件，请在导航窗格中选择 Log groups（日志组）。这些事件位于日志组 prometheus-test 中。

使用 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表安装 CloudWatch 代理

您可以使用 Amazon CloudWatch 可观测性 EKS 加载项或 Amazon CloudWatch 可观测性 Helm 图表，在 Amazon EKS 集群上安装 CloudWatch 代理和 Fluent-bit 代理。您还可以使用 Helm 图表在未托管在 Amazon EKS 上的 Kubernetes 集群上安装 CloudWatch 代理和 Fluent-bit 代理。

默认情况下，在 Amazon EKS 集群上使用任一方法均可启用具有增强 Amazon EKS 可观测性的 Container Insights 和 CloudWatch Application Signals。这两项功能都可以帮助您从集群收集基础设施指标、应用程序性能遥测和容器日志。

借助针对 Amazon EKS 增强了可观测性的 Container Insights，Container Insights 指标按每次观测收费，而不是按存储的指标或摄取的日志收费。对于 Application Signals，计费基于应用程序的入站请求、来自应用程序的出站请求以及每个配置的服务级别目标（SLO）。收到的每个入站请求都会生成一个应用程序信号，而发出的每个出站请求都会生成一个应用程序信号。每个 SLO 在每个测量周期内创建两个应用程序信号。有关 CloudWatch 定价的信息，请参阅 Amazon CloudWatch 定价。

这两种方法均可在 Amazon EKS 集群中的 Linux 和 Windows Worker 节点上启用 Container Insights。要在 Windows 上启用 Container Insights，您必须使用 Amazon EKS 加载项或 Helm 图表的 1.5.0 版或更高版本。目前，Amazon EKS 集群中的 Windows 不支持 Application Signals。

运行 Kubernetes 版本 1.23 或更高版本的 Amazon EKS 集群支持 Amazon CloudWatch Observability EKS 附加组件。

安装加载项或 Helm 图表时，还必须授予 IAM 权限，让 CloudWatch 代理能够向 CloudWatch 发送指标、日志和跟踪。有两种方式可执行此操作：

- 将策略附加到 Worker 节点的 IAM 角色。此选项会向 Worker 节点授予向 CloudWatch 发送遥测数据的权限。

- 对代理容器组（pod）的服务账户使用 IAM 角色，并将策略附加到此角色。这仅适用于 Amazon EKS 集群。此选项仅允许 CloudWatch 访问相应的代理容器组（pod）。

选项 1：使用 IAM 权限在 Worker 节点上进行安装

要使用此方法，请先输入以下命令，将 CloudWatchAgentServerPolicy IAM 策略附加到您的 Worker 节点上。在此命令中，将 *my-worker-node-role* 替换为您的 Kubernetes Worker 节点使用的 IAM 角色。

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

然后安装 Amazon CloudWatch Observability EKS 附加组件。要安装该附加组件，您可以使用 AWS CLI、控制台、AWS CloudFormation 或 Terraform。

AWS CLI

使用 AWS CLI 安装 Amazon CloudWatch Observability EKS 附加组件

输入以下 命令。将 *my-cluster-name* 替换为您的集群名称。

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-name my-cluster-name
```

Amazon EKS console

使用 Amazon EKS 控制台添加 Amazon CloudWatch Observability EKS 附加组件

1. 访问 https://console.aws.amazon.com/eks/home#/clusters 打开 Amazon EKS 控制台。
2. 在左侧导航窗格中，选择集群。
3. 选择要为其配置 Amazon CloudWatch Observability EKS 附加组件的集群名称。
4. 选择附加组件选项卡。
5. 选择获取更多附加组件。
6. 在选择附加组件页面上，执行以下操作：
   a. 在 Amazon EKS 插件部分中，选中 Amazon CloudWatch Observability 复选框。

b. 选择下一步。

7. 在配置选定的附加组件设置页面上，执行以下操作：

   a. 选择您想使用的 Version（版本）。

   b. 对于选择 IAM 角色，选择从节点继承

   c. （可选）展开可选配置设置。如果对冲突解决方法选择覆盖，则可能用 Amazon EKS 附加组件设置覆盖现有附加组件的一个或多个设置。如果不启用此选项，并且与现有设置存在冲突，则操作将失败。您可以使用生成的错误消息对冲突进行故障排除。在选择此选项之前，请确保 Amazon EKS 附加组件不会管理您需要自行管理的设置。

   d. 选择下一步。

8. 在查看和添加页面上，选择创建。附加组件安装完成后，您将看到已安装的附加组件。

AWS CloudFormation

使用 AWS CloudFormation 安装 Amazon CloudWatch Observability EKS 附加组件

将 *my-cluster-name* 替换为您的集群名称。有关更多信息，请参阅 AWS::EKS::Addon。

```
{
    "Resources": {
        "EKSAddOn": {
            "Type": "AWS::EKS::Addon",
            "Properties": {
                "AddonName": "amazon-cloudwatch-observability",
                "ClusterName": "my-cluster-name"
            }
        }
    }
}
```

Helm chart

使用 **amazon-cloudwatch-observability** Helm 图表

1. 您必须安装 Helm 才能使用此图表。有关安装 Helm 的更多信息，请参阅 Helm 文档。
2. 安装 Helm 后，输入以下命令。将 *my-cluster-name* 替换为集群的名称，并将 *my-cluster-region* 替换为集群在其中运行的区域。

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
```

```
helm repo update aws-observability
helm install --wait --create-namespace --namespace amazon-cloudwatch amazon-
cloudwatch-observability aws-observability/amazon-cloudwatch-observability --set
 clusterName=my-cluster-name --set region=my-cluster-region
```

Terraform

使用 Terraform 安装 Amazon CloudWatch Observability EKS 附加组件

将 *my-cluster-name* 替换为您的集群名称。有关更多信息，请参阅 Resource: aws_eks_addon。

```
resource "aws_eks_addon" "example" {
  addon_name   = "amazon-cloudwatch-observability"
  cluster_name = "my-cluster-name"
}
```

选项 2：使用 IAM 服务账户角色进行安装（仅适用于使用加载项的情况）

仅当您使用的是 Amazon CloudWatch 可观测性 EKS 加载项时，此方法才有效。在使用此方法之前，请验证以下先决条件：

- 在某个支持 Container Insights 的 AWS 区域 中，您具有运行正常并附加了节点的 Amazon EKS 集群。有关支持的区域列表，请参阅Container Insights。
- 您已经为集群安装并配置 kubectl。有关更多信息，请参阅 Amazon EKS 用户指南 中的安装 kubectl。
- 您已安装 eksctl。有关更多信息，请参阅《Amazon EKS 用户指南》中的安装或更新 eksctl。

使用 IAM 服务账户角色安装 Amazon CloudWatch Observability EKS 附加组件

1. 如果集群还没有 OpenID Connect（OIDC）提供程序，请输入以下命令来创建一个。有关更多信息，请参阅《Amazon EKS 用户指南》中的配置 Kubernetes 服务账户以代入 IAM 角色。

   ```
   eksctl utils associate-iam-oidc-provider --cluster my-cluster-name --approve
   ```

2. 输入以下命令以创建附加了 CloudWatchAgentServerPolicy 策略的 IAM 角色，然后使用 OIDC 将代理服务账户配置为代入该角色。将 *my-cluster-name* 替换为您的集群名称，并将 *my-*

service-account-role 替换为要将服务账户关联到的角色名称。如果角色尚不存在，eksctl 会为您创建它。

```
eksctl create iamserviceaccount \
  --name cloudwatch-agent \
  --namespace amazon-cloudwatch --cluster my-cluster-name \
  --role-name my-service-account-role \
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --role-only \
  --approve
```

3. 通过输入以下命令安装附加组件。将 *my-cluster-name* 替换为您的集群名称，将 *111122223333* 替换为账户 ID，并将 *my-service-account-role* 替换为在上一步中创建的 IAM 角色。

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-
name my-cluster-name --service-account-role-arn arn:aws:iam::111122223333:role/my-
service-account-role
```

（可选）其他配置

主题

- 选择不收集容器日志
- 使用自定义 Fluent Bit 配置
- 管理已安装容器组（pod）工作负载的 Kubernetes 容忍度
- 选择退出加速计算指标收集
- 使用自定义 CloudWatch 代理配置
- 管理准入 webhook TLS 证书
- 收集 Amazon EBS 卷 ID

选择不收集容器日志

默认情况下，该附加组件使用 Fluent Bit 从所有容器组（pod）收集容器日志，然后将日志发送到 CloudWatch Logs。有关收集哪些日志的信息，请参阅 设置 Fluent Bit。

> **Note**
>
> 加载项或 Helm 图表都不会管理集群中现有的 Fluentd 或 Fluent Bit 资源。在安装加载项或 Helm 图表之前，您可以删除现有的 Fluentd 或 Fluent Bit 资源。或者，要保留现有设置并避免加载项或 Helm 图表同时安装 Fluent Bit，您可以按照本节中的说明将其禁用。

要在使用 Amazon CloudWatch 可观测性 EKS 加载项时选择不收集容器日志，请在创建或更新加载项时传递以下选项：

```
--configuration-values '{ "containerLogs": { "enabled": false } }'
```

要在使用 Helm 图表时选择不收集容器日志，请在创建或更新加载项时传递以下选项：

```
--set containerLogs.enabled=false
```

使用自定义 Fluent Bit 配置

从 Amazon CloudWatch 可观测性 EKS 加载项的 1.7.0 版本开始，您可以在创建或更新加载项或 Helm 图表时修改 Fluent Bit 配置。您可以在加载项高级配置的 `containerLogs` 根级别部分提供自定义 Fluent Bit 配置，或者在 Helm 图表中提供覆盖值。在本节中，您可以在 `config` 部分（Linux）或 `configWindows` 部分（Windows）中提供自定义 Fluent Bit 配置。`config` 进一步细分为以下子部分：

- `service`：本部分表示用于定义 Fluent Bit 引擎全局行为的 `SERVICE` 配置。
- `customParsers`：本部分表示您想要包含的任何全局 `PARSER`，其能够获取非结构化日志条目，并为这些条目提供结构，使其更易于处理和进一步筛选。
- `extraFiles`：本部分可用于提供要包含的其他 Fluent Bit `conf` 文件。默认情况下，包含以下 3 个 `conf` 文件：
  - `application-log.conf`：用于将应用程序日志从您的集群发送到 CloudWatch Logs 中的日志组 `/aws/containerinsights/`*`my-cluster-name`*`/application` 的 `conf` 文件。
  - `dataplane-log.conf`：用于将与集群的数据面板组件（包括 CRI 日志、kubelet 日志、kube-proxy 日志和 Amazon VPC CNI 日志）对应的日志发送到 CloudWatch Logs 中的日志组 `/aws/containerinsights/`*`my-cluster-name`*`/dataplane` 的 `conf` 文件。
  - `host-log.conf`：用于将日志从 Linux 上的 `/var/log/dmesg`、`/var/log/messages`、`/var/log/secure` 以及 Windows 上的 System `winlogs` 发送到 CloudWatch Logs 中的日志组 `/aws/containerinsights/`*`my-cluster-name`*`/host` 的 `conf` 文件。

> **Note**
>
> 即使您只修改某个子部分的一个字段，也要为这些单独的部分提供完整配置。建议您使用下面提供的默认配置作为基准，然后对其进行相应的修改，这样您就不会禁用默认已启用的功能。在修改 Amazon EKS 加载项的高级配置或为 Helm 图表提供覆盖值时，您可以使用以下 YAML 配置。

要查找集群的 `config` 部分，请参阅 GitHub 上的 aws-observability / helm-charts，并找到与您安装的加载项或 Helm 图表版本相对应的版本。然后导航到 `/charts/amazon-cloudwatch-observability/values.yaml`，在 `containerLogs` 下的 `fluentBit` 部分中找到 `config` 部分（Linux）和 `configWindows` 部分（Windows）。

例如，可以在此处找到版本 1.7.0 的默认 Fluent Bit 配置。

建议您在使用 Amazon EKS 加载项的高级配置提供 YAML 或者在安装 Helm 时将其作为覆盖值提供时，将 `config` 作为 YAML 提供。确保 YAML 符合以下结构。

```
containerLogs:
  fluentBit:
    config:
      service: |
        ...
      customParsers: |
        ...
      extraFiles:
        application-log.conf: |
          ...
        dataplane-log.conf: |
          ...
        host-log.conf: |
          ...
```

以下示例 `config` 将刷新间隔的全局设置更改为 45 秒。尽管唯一的修改是对 `Flush` 字段的修改，但您仍然必须为服务子部分提供完整的 `SERVICE` 定义。由于此示例未指定其他子部分的覆盖，因此将使用默认值。

```
containerLogs:
  fluentBit:
    config:
```

```
      service: |
        [SERVICE]
          Flush                     45
          Grace                     30
          Log_Level                 error
          Daemon                    off
          Parsers_File              parsers.conf
          storage.path              /var/fluent-bit/state/flb-storage/
          storage.sync              normal
          storage.checksum          off
          storage.backlog.mem_limit 5M
```

以下示例配置包含一个额外的 Fluent bit conf 文件。在此示例中，我们在 extraFiles 下添加一个自定义 my-service.conf，它将与三个默认 extraFiles 一起包含在内。

```
containerLogs:
  fluentBit:
    config:
      extraFiles:
        my-service.conf: |
          [INPUT]
            Name              tail
            Tag               myservice.*
            Path              /var/log/containers/*myservice*.log
            DB                /var/fluent-bit/state/flb_myservice.db
            Mem_Buf_Limit     5MB
            Skip_Long_Lines   On
            Ignore_Older      1d
            Refresh_Interval  10

          [OUTPUT]
            Name                cloudwatch_logs
            Match               myservice.*
            region              ${AWS_REGION}
            log_group_name      /aws/containerinsights/${CLUSTER_NAME}/myservice
            log_stream_prefix   ${HOST_NAME}-
            auto_create_group   true
```

下一个示例将从 extraFiles 中完全删除现有的 conf 文件。该示例用空字符串覆盖 application-log.conf，从而将其完全排除在外。仅仅忽略 extraFiles 的 application-log.conf 意味着使用默认值，这不是我们在本示例中尝试实现的目标。这同样适用于删除您之前可能已添加到 extraFiles 的任何自定义 conf 文件。

```
containerLogs:
  fluentBit:
    config:
      extraFiles:
        application-log.conf: ""
```

管理已安装容器组（pod）工作负载的 Kubernetes 容忍度

从 Amazon CloudWatch 可观测性 EKS 加载项的 1.7.0 版本开始，加载项和 Helm 图表默认将 Kubernetes 容忍度设置为容忍加载项或 Helm 图表安装的容器组（pod）工作负载上的所有污点。这可确保 CloudWatch 代理和 Fluent Bit 等守护程序集默认可以在集群中的所有节点上安排容器组（pod）。有关污点和容忍度的更多信息，请参阅 Kubernetes 文档中的污点和容忍度。

加载项或 Helm 图表设置的默认容忍度如下：

```
tolerations:
- operator: Exists
```

您可以在使用加载项高级配置或安装或升级具有覆盖值的 Helm 图表时，通过在根级别设置 `tolerations` 字段来覆盖默认容忍度。示例将如下所示：

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

要完全忽略容忍度，您可以使用如下配置：

```
tolerations: []
```

对容忍度的任何更改都将应用于加载项或 Helm 图表安装的所有容器组（pod）工作负载。

选择退出加速计算指标收集

默认情况下，具有增强可观测性的 Container Insights 会收集加速计算监控的指标，包括 NVIDIA GPU 指标、AWS Trainium 和 AWS Inferentia 的 AWS Neuron 指标以及 AWS Elastic Fabric Adapter（EFA）指标。

从 EKS 加载项或 Helm 图表的版本 v1.3.0-eksbuild.1 以及 CloudWatch 代理的版本 1.300034.0 开始，默认收集来自 Amazon EKS 工作负载的 NVIDIA GPU 指标。有关收集的指标和先决条件的列表，请参阅 NVIDIA GPU 指标。

从 EKS 加载项或 Helm 图表的版本 v1.5.0-eksbuild.1 以及 CloudWatch 代理的版本 1.300036.0 开始，默认收集 AWS Trainium 和 AWS Inferentia 加速器的 AWS Neuron 指标。有关收集的指标和先决条件的列表，请参阅 AWS Trainium 和 AWS Inferentia 的 AWS Neuron 指标 。

从 EKS 加载项或 Helm 图表的版本 v1.5.2-eksbuild.1 以及 CloudWatch 代理的版本 1.300037.0 开始，默认收集来自 Amazon EKS 集群 Linux 节点的 AWS Elastic Fabric Adapter（EFA）指标。有关收集的指标和先决条件的列表，请参阅 AWS Elastic Fabric Adapter（EFA）指标 。

您可以将 CloudWatch 代理配置文件中的 accelerated_compute_metrics 字段设置为 false，从而选择不收集这些指标。此字段位于 CloudWatch 配置文件中 metrics_collected 部分的 kubernetes 部分。以下是选择不收集的配置示例。有关如何使用自定义 CloudWatch 代理配置的更多信息，请参阅以下章节：使用自定义 CloudWatch 代理配置。

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true,
        "accelerated_compute_metrics": false
      }
    }
  }
}
```

使用自定义 CloudWatch 代理配置

要使用 CloudWatch 代理收集其他指标、日志或跟踪，您可以指定自定义配置，同时保持 Container Insights 和 CloudWatch Application Signals 处于启用状态。为此，请将 CloudWatch 代理配置文件嵌入到高级配置（您可以在创建或更新 EKS 加载项或 Helm 图表时使用该配置）代理键下的配置键中。以下是您未提供任何其他配置时的默认代理配置。

> **Important**
>
> 您使用其他配置设置提供的任何自定义配置都会覆盖代理使用的默认配置。请注意不要无意中禁用默认启用的功能，例如可观测性经增强的 Container Insights 和 CloudWatch Application

Signals。在需要提供自定义代理配置的情况下，我们建议使用以下默认配置作为基准，然后对其进行相应的修改。

- 使用 Amazon CloudWatch 可观测性 EKS 加载项

```
--configuration-values '{
  "agent": {
    "config": {
      "logs": {
        "metrics_collected": {
          "application_signals": {},
          "kubernetes": {
            "enhanced_container_insights": true
          }
        }
      },
      "traces": {
        "traces_collected": {
          "application_signals": {}
        }
      }
    }

}'
```

- 使用 Helm 图表

```
--set agent.config='{
  "logs": {
    "metrics_collected": {
      "application_signals": {},
      "kubernetes": {
        "enhanced_container_insights": true
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
```

```
}'
```

以下示例显示了 Windows 上 CloudWatch 代理的默认代理配置。Windows 上的 CloudWatch 代理不支持自定义配置。

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true
      },
    }
  }
}
```

管理准入 webhook TLS 证书

Amazon CloudWatch 可观测性 EKS 加载项和 Helm 图表利用 Kubernetes 准入 webhook 来验证和更改 `AmazonCloudWatchAgent` 和 `Instrumentation` 自定义资源（CR）请求，如果启用了 CloudWatch Application Signals，还可以选择在集群上验证和更改 Kubernetes 容器组（pod）请求。在 Kubernetes 中，webhook 需要一个 TLS 证书，API 服务器配置为信任该证书，以确保安全通信。

默认情况下，Amazon CloudWatch 可观测性 EKS 加载项和 Helm 图表会自动生成自签名 CA 和由此 CA 签名的 TLS 证书，以确保 API 服务器和 webhook 服务器之间的安全通信。此自动生成的证书的默认有效期为 10 年，到期后不会自动续订。此外，每次升级或重新安装加载项或 Helm 图表时，都会重新生成 CA 捆绑包和证书，从而重置有效期。如果要更改自动生成的证书的默认有效期，则可以在创建或更新附加组件时使用以下其他配置。将 *`expiry-in-days`* 替换为所需的有效期持续时间（以天为单位）。

- 将此内容用于 Amazon CloudWatch 可观测性 EKS 加载项

```
--configuration-values '{ "admissionWebhooks": { "autoGenerateCert":
 { "expiryDays": expiry-in-days } } }'
```

- 将此内容用于 Helm 图表

```
--set admissionWebhooks.autoGenerateCert.expiryDays=expiry-in-days
```

为了获得更安全、功能更丰富的证书颁发机构解决方案，该附加组件支持选择加入 cert-manager，这是一种在 Kubernetes 中广泛采用的 TLS 证书管理解决方案，可简化获取、续订、管理和使用这些证书的过程。此解决方案可确保证书有效且最新，并尝试在证书到期前的配置时间续订证书。cert-manager 还便于从各种支持的来源（包括 AWS Certificate Manager 私有证书颁发机构）颁发证书。

我们建议您查看集群上管理 TLS 证书的最佳实践，并建议您选择使用针对生产环境的 cert-manager。请注意，如果您通过选择加入来启用 cert-manager，以便管理准入 webhook TLS 证书，则您需要在安装 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表之前，在 Amazon EKS 集群上预安装 cert-manager。有关可用安装选项的更多信息，请参阅 cert-manager 文档。安装完成后，您可以使用以下附加配置选择使用 cert-manager 来管理准入 webhook TLS 证书。

- 如果使用 Amazon CloudWatch 可观测性 EKS 加载项

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```

- 如果您使用的是 Helm 图表

```
--set admissionWebhooks.certManager.enabled=true
```


```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```


默认情况下，本节中讨论的高级配置将使用 SelfSigned 颁发者。

收集 Amazon EBS 卷 ID

如果您希望在性能日志中收集 Amazon EBS 卷 ID，则必须向附加到 Worker 节点或服务账户的 IAM 角色添加另一个策略。添加以下内容作为内联策略。有关更多信息，请参阅添加和删除 IAM 身份权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
```

```
            "Effect": "Allow"
        }
    ]
}
```

Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表问题排查

可以使用以下信息帮助对 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表的问题进行排查

主题

- 更新和删除 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表
- 验证 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表使用的 CloudWatch 代理版本
- 管理加载项或 Helm 图表时处理 ConfigurationConflict

更新和删除 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表

有关更新或删除 Amazon CloudWatch Observability EKS 附加组件的说明，请参阅管理 Amazon EKS 附加组件。使用 `amazon-cloudwatch-observability` 作为附加组件的名称。

要删除集群中的 Helm 图表，请输入以下命令。

```
helm delete amazon-cloudwatch-observability -n amazon-cloudwatch --wait
```

验证 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表使用的 CloudWatch 代理版本

Amazon CloudWatch 可观测性 EKS 加载项和 Helm 图表安装 `AmazonCloudWatchAgent` 类型的自定义资源，用于控制集群上 CloudWatch 代理守护程序集的行为，包括正在使用的 CloudWatch 代理的版本。您可以通过输入以下命令获取集群 u 上安装的所有 `AmazonCloudWatchAgent` 自定义资源的列表：

```
kubectl get amazoncloudwatchagent -A
```

在此命令的输出中，您应该能够检查 CloudWatch 代理的版本。或者，您也可以描述集群上运行的 `amazoncloudwatchagent` 资源或其中一个 `cloudwatch-agent-*` 容器组（pod）来检查正在使用的映像。

管理加载项或 Helm 图表时处理 ConfigurationConflict

在安装或更新 Amazon CloudWatch 可观测性 EKS 加载项或 Helm 图表时，如果您发现故障是由现有资源引起的，这可能是因为您已经在集群上安装了 CloudWatch 代理及其相关组件，例如 ServiceAccount、ClusterRole 和 ClusterRoleBinding。

加载项显示的错误将包括 `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`，

Helm 图表显示的错误将类似 `Error: INSTALLATION FAILED: Unable to continue with install and invalid ownership metadata.`。

当加载项或 Helm 图表尝试安装 CloudWatch 代理及其关联组件时，如果检测到内容有任何变化，在默认情况下这会使安装或更新失败，以避免覆盖集群上资源的状态。

如果您正在尝试载入 Amazon CloudWatch 可观测性 EKS 加载项，但出现此故障，则建议您删除之前在集群上安装的现有 CloudWatch 代理设置，然后安装 EKS 加载项或 Helm 图表。请务必备份您可能对原始 CloudWatch 代理设置所做的任何自定义，例如自定义代理配置，并在下次安装或更新时将其提供给加载项或 Helm 图表。如果您之前安装了 CloudWatch 代理以载入 Container Insights，请参阅 删除 Container Insights 的 CloudWatch 代理和 Fluent Bit 获取更多信息。

或者，该附加组件支持具有指定 `OVERWRITE` 功能的冲突解决配置选项。您可以使用此选项，通过覆盖集群上的冲突来继续安装或更新附加组件。如果您使用的是 Amazon EKS 控制台，则在创建或更新附加组件时选择可选配置设置时，会找到冲突解决方法。如果您使用的是 AWS CLI，则可以在命令中提供 `--resolve-conflicts OVERWRITE`，以创建或更新附加组件。

CloudWatch 代理收集的指标

您可以在服务器上安装 CloudWatch 代理以从服务器中收集指标。您可以在 Amazon EC2 实例和本地部署服务器以及运行 Linux、Windows Server 或 macOS 的电脑上安装该代理。如果在 Amazon EC2 实例上安装该代理，则除了在 Amazon EC2 实例上默认启用的指标以外，它还会收集一些其他指标。

有关在实例上安装 CloudWatch 代理的信息，请参阅 使用 CloudWatch 代理收集指标、日志和跟踪信息。

本节中讨论的所有指标均由 CloudWatch 代理直接收集。

Windows Server 实例上的 CloudWatch 代理收集的指标

通过在运行 Windows Server 的服务器上安装 CloudWatch 代理，您可以收集与 Windows 性能监视器中的计数器关联的指标。在创建这些计数器的 CloudWatch 指标名称时，将在对象名称和计数器名称

之间添加空格。例如，在 CloudWatch 中为 `Processor` 对象的 `% Interrupt Time` 计数器指定指标名称 `Processor % Interrupt Time`。有关 Windows 性能监视器计数器的更多信息，请参阅 Microsoft Windows Server 文档。

CloudWatch 代理收集的指标的默认命名空间为 `CWAgent`，不过您可以在配置该代理时指定其他命名空间。

Linux 和 macOS 实例上的 CloudWatch 代理收集的指标

下表列出了您可以使用 Linux 服务器和 macOS 电脑上的 CloudWatch 代理收集的指标。

指标	描述
`cpu_time_active`	CPU 在任何容量中处于活动状态的时间。该指标以百分之一秒为单位。 单位：无
`cpu_time_guest`	CPU 为来宾操作系统运行虚拟 CPU 的时间。该指标以百分之一秒为单位。 单位：无
`cpu_time_guest_nice`	CPU 为来宾操作系统运行虚拟 CPU 的时间，它具有较低的优先级，可能会被其他进程中断。该指标以百分之一秒为单位。 单位：无
`cpu_time_idle`	CPU 处于空闲状态的时间。该指标以百分之一秒为单位。 单位：无
`cpu_time_iowait`	CPU 等待 I/O 操作完成的时间。该指标以百分之一秒为单位。 单位：无
`cpu_time_irq`	CPU 处理中断的时间。该指标以百分之一秒为单位。

指标	描述
	单位：无
cpu_time_nice	CPU 处于用户模式的时间，它具有低优先级进程，可能很容易被高优先级进程中断。该指标以百分之一秒为单位。 单位：无
cpu_time_softirq	CPU 处理软件中断的时间。该指标以百分之一秒为单位。 单位：无
cpu_time_steal	CPU 被盗的时间，这是在虚拟化环境中的其他操作系统上所花的时间。该指标以百分之一秒为单位。 单位：无
cpu_time_system	CPU 处于系统模式的时间。该指标以百分之一秒为单位。 单位：无
cpu_time_user	CPU 处于用户模式的时间。该指标以百分之一秒为单位。 单位：无
cpu_usage_active	CPU 在任何容量中处于活动状态的时间百分比。 单位：百分比
cpu_usage_guest	CPU 为来宾操作系统运行虚拟 CPU 的时间百分比。 单位：百分比
cpu_usage_guest_nice	CPU 为来宾操作系统运行虚拟 CPU 的时间百分比，它具有较低的优先级，可能会被其他进程中断。 单位：百分比

指标	描述
cpu_usage_idle	CPU 空闲时间的百分比。 单位：百分比
cpu_usage_iowait	CPU 等待 I/O 操作完成的时间百分比。 单位：百分比
cpu_usage_irq	CPU 处理中断的时间百分比。 单位：百分比
cpu_usage_nice	CPU 处于用户模式的时间百分比，它具有低优先级进程，可能很容易被高优先级进程中断。 单位：百分比
cpu_usage_softirq	CPU 处理软件中断的时间百分比。 单位：百分比
cpu_usage_steal	CPU 被盗的时间百分比，或者在虚拟化环境中的其他操作系统上所花的时间。 单位：百分比
cpu_usage_system	CPU 处于系统模式的时间百分比。 单位：百分比
cpu_usage_user	CPU 处于用户模式的时间百分比。 单位：百分比
disk_free	磁盘上的可用空间。 单位：字节

指标	描述
disk_inodes_free	磁盘上的可用索引节点数。 单位：计数
disk_inodes_total	在磁盘上保留的总索引节点数。 单位：计数
disk_inodes_used	在磁盘上使用的索引节点数。 单位：计数
disk_total	磁盘上的总空间，包括已使用的空间和可用空间。 单位：字节
disk_used	磁盘上已使用的空间。 单位：字节
disk_used_percent	已使用总磁盘空间的百分比。 单位：百分比
diskio_iops_in_progress	已发出到设备驱动程序但尚未完成的 I/O 请求数。 单位：计数
diskio_io_time	磁盘已将 I/O 请求排队的时间。 单位：毫秒 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。

指标	描述
diskio_reads	磁盘读取操作数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
diskio_read_bytes	从磁盘中读取的字节数。 单位：字节 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
diskio_read_time	读取请求在磁盘上等待的时间。同时等待的多个读取请求会增加该数字。例如，如果 5 个请求均平均等待 100 毫秒，则报告 500。 单位：毫秒 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
diskio_writes	磁盘写入操作数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
diskio_write_bytes	写入到磁盘的字节数。 单位：字节 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。

指标	描述
diskio_write_time	写入请求在磁盘上等待的时间。同时等待的多个写入请求会增加该数字。例如，如果 8 个请求均平均等待 1000 毫秒，则报告 8000。 单位：毫秒 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
ethtool_bw_in_allowance_exceeded	因入站聚合带宽超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 metrics_collected 部分的 ethtool 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无
ethtool_bw_out_allowance_exceeded	因出站聚合带宽超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 metrics_collected 部分的 ethtool 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无

指标	描述
ethtool_conntrack_allowance_exceeded	由于连接跟踪超过实例的最大值且无法建立新连接而丢弃的数据包的数量。这可能会导致进出实例的流量丢失数据包。 仅当您将此指标列在 CloudWatch 代理配置文件 metrics_collected 部分的 ethtool 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无
ethtool_linklocal_allowance_exceeded	由于到本地代理服务的流量的 PPS 超出网络接口的最大值而丢弃的数据包数量。这会影响流向 DNS 服务、实例元数据服务和 Amazon Time Sync Service 的流量。 仅当您将此指标列在 CloudWatch 代理配置文件 metrics_collected 部分的 ethtool 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标 单位：无
ethtool_pps_allowance_exceeded	因双向 PPS 超过实例的最大值而排队和/或丢弃的数据包的数量。 仅当您将此指标列在 CloudWatch 代理配置文件 metrics_collected 部分的 ethtool 子部分时，才会收集该指标。有关更多信息，请参阅 收集网络性能指标。 单位：无
mem_active	在上一抽样周期以某种方式使用的内存量。 单位：字节

指标	描述
mem_available	可以立即分配给进程的可用内存量。 单位：字节
mem_available_percent	可以立即分配给进程的可用内存百分比。 单位：百分比
mem_buffered	用作缓冲区的内存量。 单位：字节
mem_cached	用作文件缓存的内存量。 单位：字节
mem_free	未使用的内存量。 单位：字节
mem_inactive	在上一采样周期未以某种方式使用的内存量 单位：字节
mem_total	内存的总量。 单位：字节
mem_used	当前使用的内存量。 单位：字节
mem_used_percent	当前使用的内存百分比。 单位：百分比

指标	描述
net_bytes_recv	网络接口接收的字节数。 单位：字节 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_bytes_sent	网络接口发送的字节数。 单位：字节 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_drop_in	该网络接口接收并已丢弃的数据包数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_drop_out	该网络接口发送并已丢弃的数据包数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_err_in	该网络接口检测到的接收错误数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。

指标	描述
net_err_out	该网络接口检测到的发送错误数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_packets_sent	该网络接口发送的数据包数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
net_packets_recv	该网络接口接收的数据包数。 单位：计数 应该用于该指标的唯一统计数据是 Sum。切勿使用 Average。
netstat_tcp_close	没有状态的 TCP 连接数。 单位：计数
netstat_tcp_close_wait	等待来自客户端的终止请求的 TCP 连接数。 单位：计数
netstat_tcp_closing	等待来自客户端的包含确认的终止请求的 TCP 连接数。 单位：计数
netstat_tcp_established	已建立的 TCP 连接数。 单位：计数

指标	描述
netstat_tcp_fin_wait1	在关闭连接过程中处于 FIN_WAIT1 状态的 TCP 连接数。 单位：计数
netstat_tcp_fin_wait2	在关闭连接过程中处于 FIN_WAIT2 状态的 TCP 连接数。 单位：计数
netstat_tcp_last_ack	等待客户端发送连接终止确认消息的 TCP 连接数。这是就在关闭连接之前的最后状态。 单位：计数
netstat_tcp_listen	当前侦听连接请求的 TCP 端口数。 单位：计数
netstat_tcp_none	具有不活动客户端的 TCP 连接数。 单位：计数
netstat_tcp_syn_sent	在发送连接请求后等待匹配连接请求的 TCP 连接数。 单位：计数
netstat_tcp_syn_recv	在发送和接收连接请求后等待连接请求确认的 TCP 连接数。 单位：计数
netstat_tcp_time_wait	当前等待以确保客户端收到连接终止请求确认的 TCP 连接数。 单位：计数
netstat_udp_socket	当前 UDP 连接数。 单位：计数

指标	描述
processes_blocked	已阻止的进程数。 单位：计数
processes_dead	处于“僵死”状态的进程数，它是由 Linux 上的 X 状态代码表示的。 在 macOS 电脑上不会收集此指标。 单位：计数
processes_idle	处于空闲状态 (睡眠超过 20 秒) 的进程数。仅适用于 FreeBSD 实例。 单位：计数
processes_paging	处于“分页”状态的进程数，它是由 Linux 上的 W 状态代码表示的。 在 macOS 电脑上不会收集此指标。 单位：计数
processes_running	正在运行的进程数，由 R 状态代码表示。 单位：计数
processes_sleeping	睡眠的进程数，由 S 状态代码表示。 单位：计数
processes_stopped	停止的进程数，由 T 状态代码表示。 单位：计数
processes_total	实例上的总进程数。 单位：计数

指标	描述
processes_total_threads	组成进程的总线程数。该指标仅适用于 Linux 实例。 在 macOS 电脑上不会收集此指标。 单位：计数
processes_wait	分页的进程数，它是由 FreeBSD 实例上的 W 状态代码表示的。此指标仅在 FreeBSD 实例上可用，在 Linux、Windows Server 或 macOS 实例上不可用。 单位：计数
processes_zombies	僵尸进程数，由 Z 状态代码表示。 单位：计数
swap_free	未使用的交换空间量。 单位：字节
swap_used	当前已使用的交换空间量。 单位：字节
swap_used_percent	当前已使用的交换空间百分比。 单位：百分比

CloudWatch 代理收集的内存指标的定义

当 CloudWatch 代理收集内存指标时，来源是主机的内存管理子系统。例如，Linux 内核在 /proc 中公开了操作系统维护的数据。对于内存，数据在 /proc/meminfo 中。

每种不同的操作系统和架构对进程使用的资源都有不同的计算方式。有关更多信息，请参阅以下部分。

在每个收集间隔内，每个实例上的 CloudWatch 代理会收集实例资源并计算在该实例中运行的所有进程所使用的资源。这些信息将报告回 CloudWatch 指标。您可以在 CloudWatch 代理配置文件中配置收集间隔时长。有关更多信息，请参阅 CloudWatch 代理配置文件：Agent 部分。

以下列表说明了 CloudWatch 代理收集的内存指标是如何定义的。

- 活动内存 – 进程正在使用的内存。换句话说，当前正在运行的应用程序所使用的内存。
- 可用内存 – 系统无需进入交换状态即可立即分配给进程的内存（也称为虚拟内存）。
- 缓冲内存 – 由以不同速度和优先级运行的硬件设备或程序进程共享的数据区域。
- 缓存内存 – 用于存储 CPU 接下来可能需要的程序操作中重复使用的程序指令和数据。
- 空闲内存 – 完全未使用且随时可用的内存。这部分内存完全空闲，可以在需要时供该系统使用。
- 非活动内存 –“最近”未访问过的页面。
- 总内存 – 实际物理内存 RAM 的大小。
- 已用内存 – 程序和进程当前正在使用的内存。

主题

- Linux：收集的指标和使用的计算方式
- macOS：收集的指标和使用的计算方式
- Windows：收集的指标
- 示例：在 Linux 上计算内存指标

Linux：收集的指标和使用的计算方式

收集的指标和单位：

- 活动（字节）
- 可用（字节）
- 可用百分比（百分比）
- 已缓冲（字节）
- 已缓存（字节）
- 空闲（字节）
- 非活动（字节）
- 总内存（字节）
- 已用（字节）
- 已用百分比（百分比）

已用内存 = 总内存 - 空闲内存 - 缓存内存 - 缓冲内存

总内存 = 已用内存 + 空闲内存 + 缓存内存 + 缓冲内存

macOS：收集的指标和使用的计算方式

收集的指标和单位：

- 活动（字节）
- 可用（字节）
- 可用百分比（百分比）
- 空闲（字节）
- 非活动（字节）
- 总内存（字节）
- 已用（字节）
- 已用百分比（百分比）

可用内存 = 空闲内存 + 非活动内存

已用内存 = 总内存 - 可用内存

总内存 = 可用内存 + 已用内存

Windows：收集的指标

下面列出了在 Windows 主机上收集的指标。所有这些指标的 `Unit` 都是 `None`。

- 可用字节
- 缓存中断/秒
- 页面错误/秒
- 页数/秒

没有对 Windows 指标使用任何计算方式，因为 CloudWatch 代理会解析来自性能计数器的事件。

示例：在 Linux 上计算内存指标

例如，假设在 Linux 主机上输入 **`cat /proc/meminfo`** 命令会显示以下结果：

```
MemTotal:       3824388 kB
MemFree:         462704 kB
MemAvailable:   2157328 kB
Buffers:         126268 kB
Cached:         1560520 kB
SReclaimable:    289080 kB>
```

在此示例中，CloudWatch 代理将收集以下值：CloudWatch 代理收集和报告的所有值均以字节为单位。

- `mem_total`：3916173312 字节
- `mem_available`：2209103872 字节（MemFree + Cached）
- `mem_free`：473808896 字节
- `mem_cached`：1893990400 字节（`cached` + `SReclaimable`）
- `mem_used`：1419075584 字节 (`MemTotal` – (`MemFree` + `Buffers` + (`Cached` + `SReclaimable`)))
- `mem_buffered`：129667072 字节
- `mem_available_percent`：56.41%
- `mem_used_percent`：36.24% (`mem_used` / `mem_total`) * 100

CloudWatch 代理的常见场景

以下几节简要说明如何为 CloudWatch 代理完成常见的配置和自定义任务。

主题

- 以不同用户身份运行 CloudWatch 代理
- CloudWatch 代理如何处理稀疏日志文件
- 将自定义维度添加到 CloudWatch 代理收集的指标
- 多个 CloudWatch 代理配置文件
- 汇总或累积 CloudWatch 代理收集的指标
- 使用 CloudWatch 代理收集高精度指标
- 向不同账户发送指标、日志和跟踪信息
- 统一的 CloudWatch 代理和更早的 CloudWatch Logs 代理之间的时间戳差异

以不同用户身份运行 CloudWatch 代理

在 Linux 服务器上，CloudWatch 默认以根用户身份运行。要让代理以不同用户身份运行，请在 CloudWatch 代理配置文件的 agent 部分中使用 run_as_user 参数。该选项仅在 Linux 服务器上可用。

如果已使用 root 用户运行代理并希望更改为使用不同的用户，请使用以下过程之一。

在运行 Linux 的 EC2 实例上以不同用户身份运行 CloudWatch 代理

1. 下载并安装新的 CloudWatch 代理软件包。有关更多信息，请参阅 下载 CloudWatch 代理软件包。
2. 创建新的 Linux 用户，或使用 RPM 或 DEB 文件创建的默认用户（名为 cwagent）。
3. 使用以下方式之一为该用户提供凭证：
   - 如果文件 .aws/credentials 存在于根用户的主目录中，您必须为要用于运行 CloudWatch 代理的用户创建一个凭证文件。该凭证文件是 /home/*username*/.aws/credentials。然后，将 common-config.toml 中的 shared_credential_file 参数值设置为凭证文件的路径名。有关更多信息，请参阅（可选）修改代理的通用配置或区域信息。
   - 如果根用户的主目录中不存在文件 .aws/credentials，您可以执行下列操作之一：
     - 为要用于运行 CloudWatch 代理的用户创建一个凭证文件。该凭证文件是 /home/*username*/.aws/credentials。然后，将 common-config.toml 中的 shared_credential_file 参数值设置为凭证文件的路径名。有关更多信息，请参阅（可选）修改代理的通用配置或区域信息。
     - 将 IAM 角色附加到实例，而不是创建凭证文件。代理将该角色作为凭证提供程序。
4. 在 CloudWatch 代理配置文件中，在 agent 部分中添加以下行：

   ```
   "run_as_user": "username"
   ```

   根据需要，对该配置文件进行其他修改。有关更多信息，请参阅 创建 CloudWatch 代理配置文件
5. 为用户提供所需的权限。用户必须拥有要收集的日志文件的 Read (r) 权限，并且必须对日志文件路径中的每个目录具有 Execute (x) 权限。
6. 使用刚修改的配置文件启动代理。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
   ```

在运行 Linux 的本地部署服务器上以不同用户身份运行 CloudWatch 代理

1. 下载并安装新的 CloudWatch 代理软件包。有关更多信息，请参阅 下载 CloudWatch 代理软件包。
2. 创建新的 Linux 用户，或使用 RPM 或 DEB 文件创建的默认用户（名为 cwagent）。
3. 将该用户的凭证存储到用户可访问的路径中，例如 /home/*username*/.aws/credentials。
4. 将 common-config.toml 中的 shared_credential_file 参数值设置为凭证文件的路径名。有关更多信息，请参阅 （可选）修改代理的通用配置或区域信息。
5. 在 CloudWatch 代理配置文件中，在 agent 部分中添加以下行：

   ```
   "run_as_user": "username"
   ```

   根据需要，对该配置文件进行其他修改。有关更多信息，请参阅 创建 CloudWatch 代理配置文件
6. 为用户提供所需的权限。用户必须拥有要收集的日志文件的 Read (r) 权限，并且必须对日志文件路径中的每个目录具有 Execute (x) 权限。
7. 使用刚修改的配置文件启动代理。

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
   ```

CloudWatch 代理如何处理稀疏日志文件

稀疏文件是具有空块和真实内容的文件。稀疏文件通过将表示空块的简短信息（而不是构成此块的实际 Null 字节）写入磁盘，以更有效地使用磁盘空间。这使得稀疏文件的实际大小通常比其表观大小小得多。

但是，CloudWatch 代理对稀疏文件的处理方式与对普通文件的处理方式不同。当代理读取稀疏文件时，空块被视为填充 Null 字节的“真实”块。因此，CloudWatch 代理向 CloudWatch 发布的字节数与稀疏文件的表观大小相同。

配置 CloudWatch 代理来发布稀疏文件可能会导致高于预期 CloudWatch 成本，因此我们建议不要这样做。例如，Linux 中的 /var/logs/lastlog 通常是一个非常稀疏的文件，我们建议您不要将其发布到 CloudWatch。

将自定义维度添加到 CloudWatch 代理收集的指标

要将自定义维度 (如标签) 添加到该代理收集的指标中，请将 append_dimensions 字段添加到代理配置文件中列出这些指标的部分。

例如，以下示例配置文件部分将一个名为 stackName 且值为 Prod 的自定义维度添加到该代理收集的 cpu 和 disk 指标中。

```
"cpu":{
  "resources":[
    "*"
  ],
  "measurement":[
    "cpu_usage_guest",
    "cpu_usage_nice",
    "cpu_usage_idle"
  ],
  "totalcpu":false,
  "append_dimensions":{
    "stackName":"Prod"
  }
},
"disk":{
  "resources":[
    "/",
    "/tmp"
  ],
  "measurement":[
    "total",
    "used"
  ],
  "append_dimensions":{
    "stackName":"Prod"
  }
}
```

切记，每次更改代理配置文件时，您必须重新启动该代理以使更改生效。

多个 CloudWatch 代理配置文件

在 Linux 服务器和 Windows 服务器上，您可以将 CloudWatch 代理设置为使用多个配置文件。例如，您可以使用一个常见的配置文件，该文件收集您始终要从基础设施中的所有服务器收集的一组指标、日

志和跟踪信息。然后，您可以使用其他配置文件，这些配置文件从某些应用程序或在某些情况下收集指标。

要对此进行设置，请首先创建要使用的配置文件。将在同一台服务器上一起使用的任何配置文件必须具有不同的文件名。您可以将配置文件存储在服务器或 Parameter Store 中。

使用 `fetch-config` 选项启动 CloudWatch 代理，并指定第一个配置文件。要将第二个配置文件附加到正在运行的代理，请使用相同的命令，但使用 `append-config` 选项。将收集任一配置文件中列出的所有指标、日志和跟踪信息。以下示例命令使用配置存储作为文件来演示此场景。第一行使用 `infrastructure.json` 配置文件启动代理，第二行附加 `app.json` 配置文件。

以下示例命令适用于 Linux。

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2
 -s -c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m
 ec2 -s -c file:/tmp/app.json
```

以下示例命令适用于 Windows Server。

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\infrastructure.json"
```

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a append-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\app.json"
```

以下示例配置文件展示了此功能的一个用法。第一个配置文件用于基础设施中的所有服务器，第二个配置文件仅收集来自特定应用程序的日志，并附加到运行该应用程序的服务器。

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
```

```
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
         "measurement": [
           "used_percent"
        ]
      }
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log"
          },
          {
            "file_path": "/var/log/messages",
            "log_group_name": "/var/log/messages"
          }
        ]
      }
    }
  }
}
```

app.json

```
{
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/app/app.log*",
                        "log_group_name": "/app/app.log"
                    }
```

```
                ]
            }
        }
    }
}
```

附加到配置的任何配置文件彼此之间以及与初始配置文件之间都必须具有不同的文件名。如果您将 `append-config` 用于与代理已在使用的配置文件具有相同文件名的配置文件，则附加命令将覆盖第一个配置文件中的信息，而不是附加到它。即使这两个具有相同文件名的配置文件位于不同的文件路径中，也是这样。

上述示例显示使用了两个配置文件，但对您可以附加到代理配置的配置文件数量并没有限制。您还可以混合使用位于服务器上的配置文件和位于 Parameter Store 中的配置。

汇总或累积 CloudWatch 代理收集的指标

要汇总或累积该代理收集的指标，请将 `aggregation_dimensions` 字段添加到代理配置文件中与该指标对应的部分。

例如，以下配置文件片段累积 `AutoScalingGroupName` 维度上的指标。每个 Auto Scaling 组的所有实例中的指标将进行汇总，可以将这些指标作为一个整体进行查看。

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"]]
}
```

除了累积 Auto Scaling 组名称以外，如果还希望累积各个 `InstanceId` 和 `InstanceType` 维度的组合，请添加以下内容。

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId", "InstanceType"]]
}
```

要将指标累积到一个集合中，请使用 `[]`。

```
"metrics": {
```

```
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [[]]
}
```

切记，每次更改代理配置文件时，您必须重新启动该代理以使更改生效。

使用 CloudWatch 代理收集高精度指标

metrics_collection_interval 字段指定收集的指标的时间间隔 (以秒为单位)。如果为该字段指定小于 60 的值，则将指标作为高精度指标进行收集。

例如，如果所有指标均应为高精度指标并且每 10 秒收集一次，请在 agent 部分中指定 10 作为 metrics_collection_interval 的值，以用作全局指标收集间隔。

```
"agent": {
  "metrics_collection_interval": 10
}
```

或者，以下示例将 cpu 指标设置为每秒收集一次，将所有其他指标设置为每分钟收集一次。

```
"agent":{
  "metrics_collection_interval": 60
},
"metrics":{
  "metrics_collected":{
    "cpu":{
      "resources":[
        "*"
      ],
      "measurement":[
        "cpu_usage_guest"
      ],
      "totalcpu":false,
      "metrics_collection_interval": 1
    },
    "disk":{
      "resources":[
        "/",
        "/tmp"
      ],
      "measurement":[
```

```
        "total",
        "used"
      ]
    }
  }
}
```

切记，每次更改代理配置文件时，您必须重新启动该代理以使更改生效。

向不同账户发送指标、日志和跟踪信息

要让 CloudWatch 代理将指标、日志或跟踪信息发送到其他账户，请在发送服务器上的代理配置文件中指定 `role_arn` 参数。`role_arn` 值指定在将数据发送到目标账户时代理使用的目标账户中的 IAM 角色。在指标或日志传递到目标账户时，此角色使发送账户能够在目标账户中担任相应的角色。

您还可以在代理配置文件中指定多个单独的 `role_arn` 字符串：一个用于发送指标，一个用于发送日志，还有一个用于发送跟踪信息。

配置文件的 `agent` 部分的以下部分示例将代理设置为在将数据发送到其他账户时使用 `CrossAccountAgentRole`。

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  .....
}
```

或者，以下示例为发送账户设置不同的角色，以用于发送指标、日志和跟踪信息：

```
"metrics": {
    "credentials": {
     "role_arn": "RoleToSendMetrics"
    },
    "metrics_collected": {....
```

```
"logs": {
```

```
    "credentials": {
    "role_arn": "RoleToSendLogs"
    },
    ....
```

必需策略

在代理配置文件中指定 role_arn 时，还必须确保发送和目标账户的 IAM 角色具有某些策略。发送账户和目标账户中的角色都应具有 CloudWatchAgentServerPolicy。有关将该策略分配给角色的更多信息，请参阅在 Amazon EC2 实例上创建要与 CloudWatch 代理一起使用的 IAM 角色。

发送账户中的角色还必须包含以下策略。编辑角色时，您可以将此策略添加到 IAM 控制台的 Permissions（权限）选项卡。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::target-account-ID:role/agent-role-in-target-account"
            ]
        }
    ]
}
```

目标账户中的角色必须包含以下策略，以便它识别发送账户使用的 IAM 角色。编辑角色时，您可以将此策略添加到 IAM 控制台的 True relationships（信任关系）选项卡。您添加此策略的目标账户中的角色是您在创建 IAM 角色和用户以用于 CloudWatch 代理中创建的角色。此角色是在发送账户使用的策略的 *agent-role-in-target-account* 中指定的角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::sending-account-ID:role/role-in-sender-account"
```

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

统一的 CloudWatch 代理和更早的 CloudWatch Logs 代理之间的时间戳差异

与更早的 CloudWatch Logs 代理相比，CloudWatch 代理支持将一组不同的符号集用于时间戳格式。这些差异如下表所示。

两种代理均支持的符号	仅统一 CloudWatch 代理支持的符号	仅更早的 CloudWatch Logs 代理支持的符号
%A、%a、%b、%B、%d、%f、%H、%I、%m、%M、%p、%S、%y、%Y、%Z、z	%-d、%-I、%-m、%-M、%-S	%c、%j、%U、%W、%w

有关较新的 CloudWatch 代理支持的符号的含义的更多信息，请参阅 Amazon CloudWatch 用户指南中的 CloudWatch 代理配置文件：日志部分。有关 CloudWatch Logs 代理支持的符号的信息，请参阅 Amazon CloudWatch Logs 用户指南中的代理配置文件。

CloudWatch 代理故障排除

可以使用以下信息帮助解决 CloudWatch 代理问题。

主题

- CloudWatch 代理命令行参数
- 使用 Run Command 安装 CloudWatch 代理失败
- CloudWatch 代理无法启动
- 验证 CloudWatch 代理是否正在运行
- CloudWatch 代理未启动，并且错误中提及了 Amazon EC2 区域
- 无法在 Windows Server 上启动 CloudWatch 代理

- 指标存储在何处？
- CloudWatch 代理需要很长时间才能在容器中运行或记录跃点数限制错误
- 我更新了代理配置，但在 CloudWatch 控制台中看不到新的指标或日志
- CloudWatch 代理文件和位置
- 查找有关 CloudWatch 代理版本的信息
- CloudWatch 代理生成的日志
- 停止和重新启动 CloudWatch 代理

CloudWatch 代理命令行参数

要查看 CloudWatch 代理支持的参数的完整列表，请在安装了此代理的电脑上的命令行处输入以下命令：

```
amazon-cloudwatch-agent-ctl -help
```

使用 Run Command 安装 CloudWatch 代理失败

要使用 Systems Manager Run Command 安装 CloudWatch 代理，目标服务器上的 SSM Agent 必须为 SSM Agent 代理 2.2.93.0 或更高版本。如果 SSM Agent 不是正确的版本，您可能会看到以下错误消息：

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

有关更新 SSM Agent 版本的信息，请参阅 AWS Systems Manager 用户指南中的安装和配置 SSM Agent。

CloudWatch 代理无法启动

如果 CloudWatch 代理无法启动，则您的配置可能存在问题。配置信息会记录到 `configuration-validation.log` 文件中。在 Linux 服务器上，该文件位于 `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` 中，在运行 Windows Server 的服务器上，该文件位于 `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` 中。

验证 CloudWatch 代理是否正在运行

您可以查询 CloudWatch 代理以确定它是正在运行还是已停止。可使用 AWS Systems Manager 远程执行此操作。也可以使用命令行，但仅用于检查本地服务器。

使用 Run Command 查询 CloudWatch 代理的状态

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。
4. 在 Command document (命令文档) 列表中，选择 AmazonCloudWatch-ManageAgent 旁边的按钮。
5. 在操作列表中，选择状态。
6. 对于 Optional Configuration Source (可选的配置源)，选择 default (默认值) 并将 Optional Configuration Location (可选的配置位置) 保留为空。
7. 在目标区域，选择要检查的实例。
8. 选择运行。

如果该代理正在运行，输出将类似于以下内容。

```
{
       "status": "running",
       "starttime": "2017-12-12T18:41:18",
       "version": "1.73.4"
}
```

如果该代理已停止，`"status"` 字段将显示 `"stopped"`。

使用命令行在本地查询 CloudWatch 代理的状态

- 在 Linux 服务器上，输入以下命令：

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
 status
```

在运行 Windows Server 的服务器上，以管理员身份在 PowerShell 中输入以下命令：

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
 ec2 -a status
```

CloudWatch 代理未启动，并且错误中提及了 Amazon EC2 区域

如果代理将不会启动，并且错误消息中提及了 Amazon EC2 区域端点，则您可能已将代理配置为需要访问 Amazon EC2 端点但未授予该访问权限。

例如，如果您在代理配置文件中为 `append_dimensions` 参数指定一个值（取决于 Amazon EC2 元数据），并且使用代理，则必须确保服务器能够访问 Amazon EC2 的端点。有关这些端点的更多信息，请参阅 Amazon Web Services 一般参考 中的 Amazon Elastic Compute Cloud（Amazon EC2）。

无法在 Windows Server 上启动 CloudWatch 代理

在 Windows Server 上，您可能会看到以下错误：

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be
 started due to the following
error: Cannot start service AmazonCloudWatchAgent on computer '.'.
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113
 char:12
+     $svc | Start-Service
+            ~~~~~~~~~~~~~
    + CategoryInfo          : OpenError:
 (System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
   ServiceCommandException
    + FullyQualifiedErrorId :
 CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

要修复此问题，首先确保该服务器服务正在运行。如果代理在服务器服务未运行时尝试启动，则可能看到此错误。

如果服务器服务已在运行中，可能出现以下问题。在某些 Windows Server 安装中，CloudWatch 代理需要超过 30 秒才能启动。由于默认情况下，Windows Server 的服务启动超时只有 30 秒，因此这会导致代理失败，并出现类似于以下内容的错误：

要解决此问题，请增加服务超时值。有关详细信息，请参阅服务无法启动，并在 Windows 事件日志中记录事件 7000 和 7011。

指标存储在何处？

如果 CloudWatch 代理已运行，但在 AWS Management Console 或 AWS CLI 中找不到收集的指标，请确认使用的是正确的命名空间。默认情况下，该代理收集的指标的命名空间为 `CWAgent`。您可以使用代理配置文件的 `metrics` 部分中的 `namespace` 字段自定义该命名空间。如果未看到所需的指标，请检查配置文件以确认正在使用该命名空间。

在首次下载 CloudWatch 代理软件包时，代理配置文件为 `amazon-cloudwatch-agent.json`。该文件位于运行配置向导的目录中，或者您可能已将其移到其他目录中。如果使用配置向导，该向导的代理配置文件输出命名为 `config.json`。有关配置文件（包括 `namespace` 字段）的更多信息，请参阅 CloudWatch 代理配置文件：Metrics（指标）部分。

CloudWatch 代理需要很长时间才能在容器中运行或记录跃点数限制错误

当您将 CloudWatch 代理作为容器服务运行，并希望将 Amazon EC2 指标维度添加到代理收集的所有指标时，您可能会在该代理的 v1.247354.0 版中看到以下错误：

```
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Unable to retrieve Instance
 Metadata Tags. This plugin must only be used on an EC2 instance.
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Please increase hop limit
 to 2 by following this document https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
configuring-instance-metadata-options.html#configuring-IMDS-existing-instances.
2022-06-07T03:36:11Z E! [telegraf] Error running agent: could not initialize processor
 ec2tagger: EC2MetadataRequestError: failed to get EC2 instance identity document
caused by: EC2MetadataError: failed to make EC2Metadata request
        status code: 401, request id:
caused by:
```

如果代理尝试从容器内的 IMDSv2 获取元数据，而没有适当的跃点数限制，则可能会看到此错误。在 v1.247354.0 之前的代理版本中，您可能会在没有看到日志消息的情况下遇到此问题。

要解决此问题，请按照配置实例元数据选项中的说明将跃点数限制增加到 2。

我更新了代理配置，但在 CloudWatch 控制台中看不到新的指标或日志

如果您更新 CloudWatch 代理配置文件，则下次启动代理时需要使用 **fetch-config** 选项。例如，如果您将更新的文件存储在本地计算机上，请输入以下命令：

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -s -m ec2 -c file:configuration-file-path
```

CloudWatch 代理文件和位置

下表列出了 CloudWatch 代理安装和使用的文件及其在运行 Linux 或 Windows Server 的服务器上的位置。

文件	Linux 位置	Windows Server 位置
控制代理的启动、停止和重新启动的控制脚本。	/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl 或 /usr/bin/amazon-cloudwatch-agent-ctl	$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1
代理写入的日志文件。联系 AWS Support 时，您可能需要附上此信息。	/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log 或 /var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log
代理配置验证文件。	/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log 或 /var/log/amazon/amazon-cloudwatch-ag	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log

文件	Linux 位置	Windows Server 位置
	ent/configuration-validation.log	
在向导创建后立即用于配置代理的 JSON 文件。有关更多信息，请参阅 创建 CloudWatch 代理配置文件。	/opt/aws/amazon-cloudwatch-agent/bin/config.json	$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\config.json
此 JSON 文件用于配置代理（如果此配置文件已从 Parameter Store 中下载）。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json 或 /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json
用于指定代理使用的区域和凭证信息的 TOML 文件（覆盖系统默认值）。	/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml 或 /etc/amazon/amazon-cloudwatch-agent/common-config.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml
包含转换后的 JSON 配置文件内容的 TOML 文件。amazon-cloudwatch-agent-ctl 脚本会生成此文件。用户不应直接修改此文件。它对于验证 JSON 到 TOML 的转换是否成功很有用。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml 或 /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.toml

文件	Linux 位置	Windows Server 位置
包含转换后的 JSON 配置文件内容的 YAML 文件。amazon-cloudwatch-agent-ctl 脚本会生成此文件。您不应直接修改此文件。此文件对于验证 JSON 到 YAML 的转换是否成功非常有用。	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.yaml or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.yaml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.yaml

查找有关 CloudWatch 代理版本的信息

要在 Linux 服务器上查找 CloudWatch 代理的版本号，请输入以下命令：

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

要在 Windows Server 上查找 CloudWatch 代理的版本号，请输入以下命令：

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2
 -a status
```

> **Note**
>
> 使用此命令是查找 CloudWatch 代理版本的正确方法。如果您使用控制面板中的 Program and Features（程序和功能），您看到的版本号将不是正确的版本号。

您也可以下载有关代理的最新更改的 README 文件，以及指示当前可供下载的版本号的文件。这些文件位于以下位置：

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES 或 https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/RELEASE_NOTES
- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION 或 https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/CWAGENT_VERSION

CloudWatch 代理生成的日志

该代理在运行时生成一个日志。该日志包含故障排除信息。该日志是 amazon-cloudwatch-agent.log 文件。在 Linux 服务器上，该文件位于 /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log 中，在运行 Windows Server 的服务器上，该文件位于 $Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log 中。

您可以将代理配置为在 amazon-cloudwatch-agent.log 文件中记录其他详细信息。在代理配置文件的 agent 部分中，将 debug 字段设置为 true，然后重新配置并重新启动 CloudWatch 代理。要禁用该额外信息的日志记录，请将 debug 字段设置为 false。然后重新配置并重启该代理。有关更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

在版本 1.247350.0 和更高版本的 CloudWatch 代理中，您可以选择将代理配置文件内 agent 部分中的 aws_sdk_log_level 字段设置为以下一个或多个选项。使用 | 字符分隔多个选项。

- LogDebug
- LogDebugWithSigning
- LogDebugWithHTTPBody
- LogDebugRequestRetries
- LogDebugWithEventStreamBody

有关这些选项的更多信息，请参阅 LogLevelType。

停止和重新启动 CloudWatch 代理

您可以使用 AWS Systems Manager 或命令行手动停止 CloudWatch 代理。

使用 Run Command 停止 CloudWatch 代理

1. 通过 https://console.aws.amazon.com/systems-manager/ 打开 Systems Manager 控制台。
2. 在导航窗格中，选择 Run Command。

   –或者–

   如果打开了 AWS Systems Manager 主页，请向下滚动并选择 Explore Run Command（浏览 Run Command）。
3. 选择 Run command（运行命令）。

4. 在命令文档列表中，选择 AmazonCloudWatch-ManageAgent。
5. 在 Targets（目标）区域中，选择安装了 CloudWatch 代理的实例。
6. 在操作列表中，选择停止。
7. 将 Optional Configuration Source (可选的配置源) 和 Optional Configuration Location (可选的配置位置) 保留空白。
8. 选择运行。

使用命令行在本地停止 CloudWatch 代理

- 在 Linux 服务器上，输入以下命令：

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
 stop
```

  在运行 Windows Server 的服务器上，以管理员身份在 PowerShell 中输入以下命令：

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
 ec2 -a stop
```

要重新启动该代理，请按照（可选）修改 CloudWatch 代理的通用配置和命名配置文件中的说明进行操作。

在日志中嵌入指标

利用 CloudWatch 嵌入式指标格式，您可以通过写入 CloudWatch Logs 的日志形式来异步生成自定义指标。您可以将自定义指标与详细的日志事件数据一同嵌入，CloudWatch 会自动摄取自定义指标，以便您可以对其进行可视化和设置告警，从而执行实时事件检测。此外，可以使用 CloudWatch Logs Insights 查询与摄取的指标相关的详细日志事件，以提供对操作事件根本原因的深入洞察。

嵌入式指标格式可帮助您通过临时资源（例如 Lambda 函数和容器）生成可操作的自定义指标。通过使用嵌入式指标格式发送来自这些临时资源的日志，您现在可以轻松创建自定义指标，而无需检测或维护单独的代码，同时获得对日志数据的强大分析功能。

无需设置即可使用嵌入式指标格式。您可以按照 嵌入式指标格式规范 来构建日志，也可以使用我们的客户端库生成日志，然后使用 PutLogEvents API 或 CloudWatch 代理 将其发送到 CloudWatch Logs。

日志提取和存档以及生成的自定义指标会产生费用。有关更多信息，请参阅 Amazon CloudWatch 定价。

> **Note**
>
> 配置指标提取时请务必小心，因为它会影响自定义指标的使用和相应的账单。如果您无意中创建了基于高基数维度的指标（例如 `requestId`），则嵌入式指标格式将通过设计创建与每个唯一维度组合对应的自定义指标。有关更多信息，请参阅维度。

主题

- 发布使用嵌入式指标格式的日志
- 在控制台中查看您的指标和日志
- 为使用嵌入式指标格式创建的指标设置警报

发布使用嵌入式指标格式的日志

您可以使用以下方法生成嵌入式指标格式日志：

- 使用 开源客户端库 生成和发送日志。
- 按照 嵌入式指标格式规范 手动生成日志，然后使用 CloudWatch 代理 或 PutLogEvents API 发送日志。

主题

- 使用客户端库创建使用嵌入式指标格式的日志
- 规范：嵌入式指标格式
- 使用 PutLogEvents API 发送手动创建的嵌入式指标格式日志
- 使用 CloudWatch 代理发送嵌入式指标格式日志
- 通过 AWS Distro for OpenTelemetry 使用嵌入式指标格式

使用客户端库创建使用嵌入式指标格式的日志

Amazon 提供了开源客户端库，可供您用来创建嵌入式指标格式日志。目前，这些库提供以下列表中的语言版本。有关具有不同设置的完整示例，请访问我们客户端库中的 /examples。

库及其使用说明位于 Github 上。请使用以下链接。

- Node.Js

> **Note**
>
> 对于 Node.js，需要使用 4.1.1+、3.0.2+、2.0.7+ 版本与 Lambda JSON 日志格式搭配使用。在这类 Lambda 环境中使用之前的版本会导致指标丢失。
> 有关更多信息，请参阅访问 AWS Lambda 的 Amazon CloudWatch Logs。

- Python
- Java
- C#

客户端库旨在可按开箱即用的方式与 CloudWatch 代理配合使用。生成的嵌入式指标格式日志将发送到 CloudWatch 代理，然后由代理汇总并为您发布到 CloudWatch Logs。

> **Note**
>
> 使用 Lambda 时，无需代理即可将日志发送到 CloudWatch。任何记录到 STDOUT 的日志都会通过 Lambda 日志代理发送到 CloudWatch Logs。

规范：嵌入式指标格式

CloudWatch 嵌入式指标格式是一个 JSON 规范，用于指示 CloudWatch Logs 自动提取结构化日志事件中嵌入的指标值。您可以使用 CloudWatch 基于提取的指标值绘制图表并创建告警。

嵌入式指标格式规范约定

此格式规范中的关键字“MUST”、“MUST NOT”、“REQUIRED”、“SHALL”、“SHALL NOT”、“SHOULD”、“SHOULD NOT”、“RECOMMENDED”、“MAY”和“OPTIONAL”的解释如关键字 RFC2119 中所述。

此格式规范中的属于“JSON”、“JSON text”、“JSON value”、“member”、“element”、“object”、“array”、“number”、“string”、“boolean”、“true”、“false”和“null”的解释如 JavaScript 对象表示法 RFC8259 中所述。

> **Note**
>
> 如果您计划为使用嵌入式指标格式创建的指标创建告警，请参阅 为使用嵌入式指标格式创建的指标设置警报 获取相关建议。

嵌入式指标格式文档结构

此节介绍了嵌入式指标格式文档的结构。嵌入式指标格式文档用 JavaScript 对象表示法 RFC8259 进行定义。

除非另有说明，否则此规范定义的对象不得包含任何其他成员。必须忽略此规范无法识别的成员。此规范中定义的成员是区分大小写的。

嵌入式指标格式受与标准 CloudWatch Logs 事件相同的限制，并且其最大大小限制为 256KB。

通过嵌入式指标格式，您可以按指标跟踪发布在您账户的 `AWS/Logs` 命名空间中的 EMF 日志的处理。可通过这些日志跟踪 EMF 中指标生成失败的情况，以及失败的原因是解析还是验证。有关更多详细信息，请参阅使用 CloudWatch 指标监控。

根节点

LogEvent 消息必须是一个有效的 JSON 对象，并且 LogEvent 消息字符串的开头或结尾没有其他数据。有关 LogEvent 结构的更多信息，请参阅 InputLogEvent。

嵌入式指标格式文档必须包含根节点上的以下顶级成员。这是一个 元数据对象 对象。

```
{
 "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

根节点必须包含由 MetricDirective 对象 中的引用定义的所有 目标成员 成员。

根节点可以包含上述要求中未包含的任何其他成员。这些成员的值必须是有效的 JSON 类型。

元数据对象

_aws 成员可用于表示有关负载的元数据，告知下游服务应如何处理 LogEvent。该值必须是一个对象，并且必须包含以下成员：

- CloudWatchMetrics – 一个 MetricDirective 对象 数组，用于指示 CloudWatch 从 LogEvent 的根节点提取指标。

  ```
  {
    "_aws": {
      "CloudWatchMetrics": [ ... ]
    }
  }
  ```

- 时间戳 – 一个数字，表示用于从事件中提取的指标的时间戳。值必须表示为自 1970 年 1 月 1 日 00:00:00 UTC 以来的毫秒数。

  ```
  {
    "_aws": {
      "Timestamp": 1559748430481
    }
  }
  ```

MetricDirective 对象

MetricDirective 对象指示下游服务 LogEvent 包含将提取并发布到 CloudWatch 的指标。MetricDirectives 必须包含以下成员：

- 命名空间 – 一个表示指标的 CloudWatch 命名空间的字符串。

- 维度 – 一个 DimensionSet 数组。
- 指标 – 一个 MetricDefinition 对象数组。此数组不得包含 100 个以上的 MetricDefinition 对象。

DimensionSet 数组

DimensionSet 是一个字符串数组，其中包含将应用于文档中的所有指标的维度键。此数组中的值也必须是根节点上的成员 - 称为 目标成员

DimensionSet 不得包含 30 个以上的维度键。DimensionSet 可以为空。

目标成员必须具有字符串值。此值不得包含 1024 个以上的字符。目标成员定义了将作为指标标识的一部分发布的维度。使用的每个 DimensionSet 都会在 CloudWatch 中创建一个新指标。有关维度的更多信息，请参阅维度和维度。

```
{
 "_aws": {
   "CloudWatchMetrics": [
     {
       "Dimensions": [ [ "functionVersion" ] ],
       ...
     }
   ]
 },
 "functionVersion": "$LATEST"
}
```

> **Note**
>
> 配置指标提取时请务必小心，因为它会影响自定义指标的使用和相应的账单。如果您无意中创建了基于高基数维度的指标（例如 `requestId`），则嵌入式指标格式将通过设计创建与每个唯一维度组合对应的自定义指标。有关更多信息，请参阅维度。

MetricDefinition 对象

MetricDefinition 是一个必须包含以下成员的对象：

- 名称 – 指标 目标成员 的字符串 参考值。指标目标必须是数值或数值数组。

MetricDefinition 对象可以包含以下成员：

- 单位 – 一个可选字符串值，表示相应指标的度量单位。值应是有效的 CloudWatch 指标单位。有关有效单位的信息，请参阅 MetricDatum。如果未提供值，则假定默认值为 NONE。
- StorageResolution – 可选整数值，表示相应指标的存储分辨率。将此值设置为 1，这样会将此指标指定为高分辨率指标，以便 CloudWatch 以亚分钟分辨率将指标存储到 1 秒。将此值设置为 60，这样会将此指标指定为标准分辨率，CloudWatch 以 1 分钟分辨率存储该指标。值应是支持 CloudWatch 的有效分辨率，即 1 或 60。如果未提供值，则假定默认值为 60。

  有关高精度指标的更多信息，请参阅 高精度指标。

> **Note**
>
> 如果您计划为使用嵌入式指标格式创建的指标创建告警，请参阅 为使用嵌入式指标格式创建的指标设置警报 获取相关建议。

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ],
        ...
      }
    ]
  },
  "Time": 1
}
```

参考值

参考值是引用根节点上的 目标成员 成员的字符串值。这些引用不应与 RFC6901 中描述的 JSON 指针混淆。无法嵌套目标值。

目标成员

有效目标必须是根节点上的成员，而不能是嵌套对象。例如，"A.a" 的参考值必须与以下成员匹配：

```
{ "A.a" }
```

它不得与嵌套成员匹配：

```
{ "A": { "a" } }
```

目标成员的有效值取决于引用它们的对象。指标目标必须是数值或数值数组。数字数组指标目标的成员不得超过 100 个。维度目标必须具有字符串值。

嵌入式指标格式示例和 JSON 架构

以下是嵌入式指标格式的有效示例。

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ]
      }
    ]
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

您可以使用以下架构验证嵌入式指标格式文档。

```
{
```

```
    "type": "object",
    "title": "Root Node",
    "required": [
        "_aws"
    ],
    "properties": {
        "_aws": {
            "$id": "#/properties/_aws",
            "type": "object",
            "title": "Metadata",
            "required": [
                "Timestamp",
                "CloudWatchMetrics"
            ],
            "properties": {
                "Timestamp": {
                    "$id": "#/properties/_aws/properties/Timestamp",
                    "type": "integer",
                    "title": "The Timestamp Schema",
                    "examples": [
                        1565375354953
                    ]
                },
                "CloudWatchMetrics": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics",
                    "type": "array",
                    "title": "MetricDirectives",
                    "items": {
                        "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
                        "type": "object",
                        "title": "MetricDirective",
                        "required": [
                            "Namespace",
                            "Dimensions",
                            "Metrics"
                        ],
                        "properties": {
                            "Namespace": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Namespace",
                                "type": "string",
                                "title": "CloudWatch Metrics Namespace",
                                "examples": [
                                    "MyApp"
```

```
                                ],
                                "pattern": "^(.*)$",
                                "minLength": 1,
                                "maxLength": 1024
                            },
                            "Dimensions": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
                                "type": "array",
                                "title": "The Dimensions Schema",
                                "minItems": 1,
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items",
                                    "type": "array",
                                    "title": "DimensionSet",
                                    "minItems": 0,
                                    "maxItems": 30,
                                    "items": {
                                        "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                                        "type": "string",
                                        "title": "DimensionReference",
                                        "examples": [
                                            "Operation"
                                        ],
                                        "pattern": "^(.*)$",
                                        "minLength": 1,
                                        "maxLength": 250
}
                                }
                            },
                            "Metrics": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
                                "type": "array",
                                "title": "MetricDefinitions",
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items",
                                    "type": "object",
                                    "title": "MetricDefinition",
                                    "required": [
                                        "Name"
```

```
                                    ],
                                    "properties": {
                                        "Name": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Name",
                                            "type": "string",
                                            "title": "MetricName",
                                            "examples": [
                                                "ProcessingLatency"
                                            ],
                                            "pattern": "^(.*)$",
                                            "minLength": 1,
                                            "maxLength": 1024
                                        },
                                        "Unit": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Unit",
                                            "type": "string",
                                            "title": "MetricUnit",
                                            "examples": [
                                                "Milliseconds"
                                            ],
                                            "pattern": "^(Seconds|Microseconds|
Milliseconds|Bytes|Kilobytes|Megabytes|Gigabytes|Terabytes|Bits|Kilobits|Megabits|
Gigabits|Terabits|Percent|Count|Bytes\\/Second|Kilobytes\\/Second|Megabytes\\/Second|
Gigabytes\\/Second|Terabytes\\/Second|Bits\\/Second|Kilobits\\/Second|Megabits\\/
Second|Gigabits\\/Second|Terabits\\/Second|Count\\/Second|None)$"
                                        },
                                        "StorageResolution": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/StorageResolution",
                                            "type": "integer",
                                            "title": "StorageResolution",
                                            "examples": [
                                                60
                                            ]
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
```

```
        }
    }
}
```

使用 PutLogEvents API 发送手动创建的嵌入式指标格式日志

您可以使用 CloudWatch Logs PutLogEvents API 将嵌入式指标格式日志发送到 CloudWatch Logs。在调用 PutLogEvents 时，您可以选择包含以下 HTTP 标头以指示 CloudWatch Logs 应提取指标，但不再需要。

```
x-amzn-logs-format: json/emf
```

以下是使用 AWS SDK for Java 2.x 的完整示例：

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
        public static void main(String[] args) {

                final String usage = "To run this example, supply a Region code (eg.
 us-east-1), log group, and stream name as command line arguments"
                                + "Ex: PutLogEvents <region-id> <log-group-name>
 <stream-name>";

                if (args.length != 3) {
                        System.out.println(usage);
                        System.exit(1);
                }

                String regionId = args[0];
                String logGroupName = args[1];
                String logStreamName = args[2];
```

```
                CloudWatchLogsClient logsClient =
 CloudWatchLogsClient.builder().region(Region.of(regionId)).build();

                // Build a JSON log using the EmbeddedMetricFormat.
                long timestamp = System.currentTimeMillis();
                String message = "{" +
                                 "  \"_aws\": {" +
                                 "    \"Timestamp\": " + timestamp  + "," +
                                 "    \"CloudWatchMetrics\": [" +
                                 "      {" +
                                 "        \"Namespace\": \"MyApp\"," +
                                 "        \"Dimensions\": [[\"Operation\"], [\"Operation
\", \"Cell\"]]," +
                                 "        \"Metrics\": [{ \"Name\": \"ProcessingLatency
\", \"Unit\": \"Milliseconds\", \"StorageResolution\": 60 }]" +
                                 "      }" +
                                 "    ]" +
                                 "  }," +
                                 "  \"Operation\": \"Aggregator\"," +
                                 "  \"Cell\": \"001\"," +
                                 "  \"ProcessingLatency\": 100" +
                                 "}";
                InputLogEvent inputLogEvent = InputLogEvent.builder()
                        .message(message)
                        .timestamp(timestamp)
                        .build();

                // Specify the request parameters.
                PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
                        .logEvents(Collections.singletonList(inputLogEvent))
                        .logGroupName(logGroupName)
                        .logStreamName(logStreamName)
                        .build();

                logsClient.putLogEvents(putLogEventsRequest);

                System.out.println("Successfully put CloudWatch log event");
        }

}
```

> **Note**
>
> 通过嵌入式指标格式，您可以按指标跟踪发布在您账户的 AWS/Logs 命名空间中的 EMF 日志的处理。可通过这些日志跟踪 EMF 中指标生成失败的情况，以及失败的原因是解析还是验证。有关更多详细信息，请参阅使用 CloudWatch 指标监控。

使用 CloudWatch 代理发送嵌入式指标格式日志

要使用此方法，请先安装要从中发送嵌入式指标格式日志的服务的 CloudWatch 代理，然后才能开始发送事件。

CloudWatch 代理必须是 1.230621.0 版或更高版本。

> **Note**
>
> 您无需安装 CloudWatch 代理即可通过 Lambda 函数发送日志。
> 不会自动处理 Lambda 函数超时。这意味着，如果您的函数在指标刷新前超时，则不会捕获该调用的指标。

安装 CloudWatch 代理

为要将嵌入式指标格式日志发送到的每项服务安装 CloudWatch 代理。

在 EC2 上安装 CloudWatch 代理

首先，在实例上安装 CloudWatch 代理。有关更多信息，请参阅安装 CloudWatch 代理。

安装该代理后，请将它配置为侦听嵌入式指标格式日志的 UDP 或 TCP 端口。以下是侦听默认套接字 tcp:25888 的此配置的示例。有关代理配置的更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
```

```
}
```

在 Amazon ECS 上安装 CloudWatch 代理

在 Amazon ECS 上部署 CloudWatch 代理的最简单方法是将它作为附加项运行，并在与您的应用程序相同的任务定义中定义它。

创建代理配置文件

本地创建 CloudWatch 代理配置文件。在此示例中，相对文件路径将是 `amazon-cloudwatch-agent.json`。

有关代理配置的更多信息，请参阅 手动创建或编辑 CloudWatch 代理配置文件。

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

将配置推送到 SSM Parameter Store

输入以下命令以将 CloudWatch 代理配置文件推送到 AWS Systems Manager (SSM) Parameter Store。

```
aws ssm put-parameter \
    --name "cwagentconfig" \
    --type "String" \
    --value "`cat amazon-cloudwatch-agent.json`" \
    --region "{{region}}"
```

配置任务定义

配置任务定义以使用 CloudWatch 代理并开放 TCP 或 UDP 端口。应使用的示例任务定义取决于您的联网模式。

请注意，`webapp` 指定了 `AWS_EMF_AGENT_ENDPOINT` 环境变量。它将由库使用，并且应指向代理正在侦听的终端节点。此外，`cwagent` 指定 `CW_CONFIG_CONTENT` 作为“valueFrom”参数，该参数指向您在上一步中创建的 SSM 配置。

本节包含一个桥式模式示例和一个主机或 awsvpc 模式示例。有关如何在 Amazon ECS 上配置 CloudWatch 代理的更多示例，请参阅 GitHub 示例存储库

以下是桥接模式的示例。在启用桥式模式联网后，需要使用 `links` 参数将代理链接到您的应用程序，并且必须使用容器名称进行寻址。

```
{
  "containerDefinitions": [
          {
              "name": "webapp",
              "links": [ "cwagent" ],
              "image": "my-org/web-app:latest",
              "memory": 256,
              "cpu": 256,
              "environment": [{
                "name": "AWS_EMF_AGENT_ENDPOINT",
                "value": "tcp://cwagent:25888"
              }],
          },
          {
              "name": "cwagent",
              "mountPoints": [],
              "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
              "memory": 256,
              "cpu": 256,
              "portMappings": [{
                  "protocol": "tcp",
                  "containerPort": 25888
              }],
              "environment": [{
                "name": "CW_CONFIG_CONTENT",
                "valueFrom": "cwagentconfig"
              }],
          }
      ],
}
```

以下是主机模式或 awsvpc 模式的示例。在这些网络模式上运行时，可以通过 `localhost` 对代理进行寻址。

```
{
  "containerDefinitions": [
```

```
            {
                "name": "webapp",
                "image": "my-org/web-app:latest",
                "memory": 256,
                "cpu": 256,
                "environment": [{
                  "name": "AWS_EMF_AGENT_ENDPOINT",
                  "value": "tcp://127.0.0.1:25888"
                }],
            },
            {
                "name": "cwagent",
                "mountPoints": [],
                "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
                "memory": 256,
                "cpu": 256,
                "portMappings": [{
                    "protocol": "tcp",
                    "containerPort": 25888
                }],
                "environment": [{
                  "name": "CW_CONFIG_CONTENT",
                  "valueFrom": "cwagentconfig"
                }],
            }
        ],
}
```

Note

在 awsvpc 模式中，您必须为 VPC 提供公有 IP 地址（仅限 Fargate）、设置 NAT 网关或设置 CloudWatch Logs VPC 终端节点。有关设置 NAT 的更多信息，请参阅 NAT 网关。有关设置 CloudWatch Logs VPC 终端节点的更多信息，请参阅将 CloudWatch Logs 与接口 VPC 终端节点结合使用。

以下是如何将公有 IP 地址分配给使用 Fargate 启动类型的任务的示例。

```
aws ecs run-task \
--cluster {{cluster-name}} \
--task-definition cwagent-fargate \
--region {{region}} \
--launch-type FARGATE \
```

```
--network-configuration
 "awsvpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENA
```

确保权限

确保执行任务的 IAM 角色具有从 SSM Parameter Store 中进行读取的权限。您可以通过附加 AmazonSSMReadOnlyAccess 策略来添加此权限。为此，请输入以下命令。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess
 \
--role-name CWAgentECSExecutionRole
```

在 Amazon EKS 上安装 CloudWatch 代理

如果您已在此集群上安装 CloudWatch Container Insights，则可以跳过此过程的某些部分。

权限

如果您尚未安装 Container Insights，请先确保您的 Amazon EKS 节点具有适当的 IAM 权限。它们应已附加 CloudWatchAgentServerPolicy。有关更多信息，请参阅验证 先决条件。

创建 ConfigMap

为代理创建 ConfigMap。ConfigMap 还将告知代理侦听 TCP 或 UDP 端口。使用以下 ConfigMap。

```
# cwagent-emf-configmap.yaml
apiVersion: v1
data:
  # Any changes here must not break the JSON format
  cwagentconfig.json: |
    {
      "agent": {
        "omit_hostname": true
      },
      "logs": {
        "metrics_collected": {
          "emf": { }
        }
      }
    }
kind: ConfigMap
metadata:
```

```
  name: cwagentemfconfig
  namespace: default
```

如果您已安装 Container Insights，请将以下 "emf": { } 行添加到现有 ConfigMap。

应用 ConfigMap

输入以下命令可应用 ConfigMap。

```
kubectl apply -f cwagent-emf-configmap.yaml
```

部署代理

要将 CloudWatch 代理部署为附加项，请将代理添加到 pod 定义中，如以下示例所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
    - name: web-app
      image: my-org/web-app:latest
    # CloudWatch Agent configuration
    - name: cloudwatch-agent
      image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
      imagePullPolicy: Always
      resources:
        limits:
          cpu: 200m
          memory: 100Mi
        requests:
          cpu: 200m
          memory: 100Mi
      volumeMounts:
        - name: cwagentconfig
          mountPath: /etc/cwagentconfig
      ports:
  # this should match the port configured in the ConfigMap
        - protocol: TCP
          hostPort: 25888
```

```
          containerPort: 25888
  volumes:
    - name: cwagentconfig
      configMap:
        name: cwagentemfconfig
```

使用 CloudWatch 代理发送嵌入式指标格式日志

在安装并运行 CloudWatch 代理后，您可以通过 TCP 或 UDP 发送嵌入式指标格式日志。在通过代理发送日志时有两个要求：

- 日志必须包含一个 LogGroupName 键，该键告知代理要使用的日志组。
- 每个日志事件必须位于单个行上。换句话说，日志事件不能包含换行符 (\n)。

日志事件还必须遵循嵌入式指标格式规范。有关更多信息，请参阅规范：嵌入式指标格式 。

如果您计划为使用嵌入式指标格式创建的指标创建告警，请参阅 为使用嵌入式指标格式创建的指标设置警报 获取相关建议。

以下是手动从 Linux Bash Shell 发送日志事件的示例。您可以改用所选编程语言提供的 UDP 套接字接口。

```
echo '{"_aws":{"Timestamp":1574109732004,"LogGroupName":"Foo","CloudWatchMetrics":
[{"Namespace":"MyApp","Dimensions":[["Operation"]],"Metrics":
[{"Name":"ProcessingLatency","Unit":"Milliseconds","StorageResolution":60}]}]},"Operation":"Agg
 \
> /dev/udp/0.0.0.0/25888
```

> **Note**
>
> 通过嵌入式指标格式，您可以按指标跟踪发布在您账户的 AWS/Logs 命名空间中的 EMF 日志的处理。可通过这些日志跟踪 EMF 中指标生成失败的情况，以及失败的原因是解析还是验证。有关更多详细信息，请参阅使用 CloudWatch 指标监控。

通过 AWS Distro for OpenTelemetry 使用嵌入式指标格式

您可以将嵌入式指标格式用作 OpenTelemetry 项目的一部分。OpenTelemetry 是一项开源计划，通过提供一组规范和 API，消除供应商的跟踪、日志和指标特定格式之间的界限和限制。有关更多信息，请参阅 OpenTelemetry。

通过 OpenTelemetry 使用嵌入式指标格式需要两个组件：启用 OpenTelemetry 合规的数据源和 AWS Distro for OpenTelemetry Collector，以与 CloudWatch 嵌入式指标格式日志一起使用。

我们已对由 AWS 维护的 OpenTelemetry 组件的重新分配进行预先配置，以便尽可能简易地开展引导。有关使用带嵌入式指标格式的 OpenTelemetry 以及其他 AWS 服务的更多信息，请参阅 AWS Distro for OpenTelemetry。

有关语言支持和使用情况的其他信息，请参阅 Github 上的 AWS 观察。

在控制台中查看您的指标和日志

在生成用于提取指标的嵌入式指标格式日志后，可以使用 CloudWatch 控制台查看指标。嵌入式指标具有您在生成日志时指定的维度。此外，您使用客户端库生成的嵌入式指标将具有以下默认维度：

- ServiceType
- ServiceName
- LogGroup

查看从嵌入式指标格式日志生成的指标

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，请选择指标。
3. 选择在生成嵌入式指标时为其指定的命名空间。如果您使用客户端库生成指标，但未指定命名空间，请选择 aws-embedded-metrics。这是使用客户端库生成的嵌入式指标的默认命名空间。
4. 选择指标维度（例如，ServiceName）。
5. All metrics 选项卡显示命名空间中该维度的所有指标。您可执行以下操作：
    a. 要对表进行排序，请使用列标题。
    b. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。
    c. 要按资源进行筛选，请选择资源 ID，然后选择 Add to search。
    d. 要按指标进行筛选，请选择指标名称，然后选择 Add to search。

使用 CloudWatch Logs Insights 查询日志

您可以使用 CloudWatch Logs Insights 查询与提取的指标相关的详细日志事件，以提供对操作事件根本原因的深入洞察。从日志中提取指标的好处之一是，您可以稍后按独特指标（指标名称加独特维度集）和指标值筛选日志，以获取生成聚合指标值的事件的上下文

例如，要获取受影响的请求 ID 或 X-ray 跟踪 ID，您可以在 CloudWatch Logs Insights 中运行以下查询。

```
filter Latency > 1000 and Operation = "Aggregator"
| fields RequestId, TraceId
```

您还可以对高基数键执行查询时间聚合，例如查找受事件影响的客户。以下示例对此进行了说明。

```
filter Latency > 1000 and Operation = "Aggregator"
| stats count() by CustomerId
```

有关更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据

为使用嵌入式指标格式创建的指标设置警报

通常，为使用嵌入式指标格式生成的指标创建告警的模式与为任何其他指标创建告警的模式相同。有关更多信息，请参阅 使用 Amazon CloudWatch 告警。

嵌入式指标格式指标的生成，具体取决于您的日志发布流程，因为日志必须由 CloudWatch Logs 处理后才能转换为指标。因此，及时发布日志对您来说非常重要，这样您的指标数据点才能在评估告警的时间段内创建。

如果您计划使用嵌入式指标格式发送高分辨率指标并为这些指标创建告警，我们建议您每隔 5 秒或更短的时间将日志刷新到 CloudWatch Logs，以避免引入额外的延迟，从而导致部分或丢失的数据出现告警。如果您使用的是 CloudWatch 代理，则可以通过在 CloudWatch 代理配置文件中设置 `force_flush_interval` 参数来调整刷新间隔。此值默认为 5 秒。

如果您在无法控制日志刷新间隔的其他平台上使用 Lambda，请考虑使用“M out of N”告警来控制用于告警的数据点数量。有关更多信息，请参阅评估告警。

发布 CloudWatch 指标的 AWS 服务

以下 AWS 服务将指标发布到 CloudWatch。有关指标和维度的信息，请参阅指定的文档。

服务	命名空间	文档
AWS Amplify	AWS/AmplifyHosting	监控
Amazon API Gateway	AWS/ApiGateway	使用 Amazon CloudWatch 监控 API 执行
Amazon AppFlow	AWS/AppFlow	使用 Amazon CloudWatch 监控 Amazon AppFlow
AWS Application Migration Service	AWS/MGN	使用 Amazon CloudWatch 监控应用程序迁移服务
AWS App Runner	AWS/AppRunner	查看报告给 CloudWatch 的 App Runner 服务指标
AppStream 2.0	AWS/AppStream	监控 Amazon AppStream 2.0 资源
AWS AppSync	AWS/AppSync	CloudWatch 指标
Amazon Athena	AWS/Athena	使用 CloudWatch 指标监控 Athena 查询
Amazon Aurora	AWS/RDS	Amazon Aurora 指标
AWS Backup	AWS/Backup	使用 CloudWatch 监控 AWS 备份指标
Amazon Bedrock	AWS/Bedrock	使用 Amazon CloudWatch 监控 Amazon Bedrock
AWS Billing and Cost Management	AWS/Billing	使用提醒和通知监控费用

服务	命名空间	文档
Amazon Braket	AWS/Braket/By Device	使用 Amazon CloudWatch 监控 Amazon Braket
AWS Certificate Manager	AWS/CertificateManager	支持的 CloudWatch 指标
AWS 私有 CA	AWS/ACMPrivateCA	支持的 CloudWatch 指标
AWS Chatbot	AWS/Chatbot	使用 Amazon CloudWatch 监控 AWS Chatbot
Amazon Chime	AWS/ChimeVoiceConnector	使用 Amazon CloudWatch 监控 Amazon Chime
Amazon Chime SDK	AWS/ChimeSDK	服务指标
AWS Client VPN	AWS/ClientVPN	使用 Amazon CloudWatch 进行监控
Amazon CloudFront	AWS/CloudFront	使用 CloudWatch 监控 CloudFront 活动
AWS CloudHSM	AWS/CloudHSM	获取 CloudWatch 指标
Amazon CloudSearch	AWS/CloudSearch	使用 Amazon CloudWatch 监控 Amazon CloudSearch 域
AWS CloudTrail	AWS/CloudTrail	支持的 CloudWatch 指标
CloudWatch 代理	CWAgent 或者自定义命名空间	CloudWatch 代理收集的指标

服务	命名空间	文档
CloudWatch Application Signals	ApplicationSignals	收集的标准应用程序指标
CloudWatch 指标流	AWS/CloudWatch/MetricStreams	使用 CloudWatch 指标监控您的指标流
CloudWatch RUM	AWS/RUM	您可以使用 CloudWatch RUM 收集的 CloudWatch 指标
CloudWatch Synthetics	CloudWatchSynthetics	金丝雀发布的 CloudWatch 指标
Amazon CloudWatch Logs	AWS/Logs	使用 CloudWatch 指标监控使用情况
AWS CodeBuild	AWS/CodeBuild	监控 AWS CodeBuild
Amazon CodeGuru Reviewer		使用 Amazon CloudWatch 监控 CodeGuru Reviewer
Amazon Kendra		使用 Amazon CloudWatch 监控 Amazon Kendra
Amazon Q 开发者版	AWS/CodeWhisperer	使用 Amazon CloudWatch 监控 Amazon Q 开发者版
Amazon Cognito	AWS/Cognito	监控 Amazon Cognito
Amazon Comprehend	AWS/Comprehend	监控 Amazon Comprehend 端点
AWS Config	AWS/Config	AWS Config 用量和成功指标

服务	命名空间	文档
Amazon Connect	AWS/Connect	使用 Amazon CloudWatch 指标监控 Amazon Connect
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	使用 Amazon CloudWatch 监控您的策略
AWS DataSync	AWS/DataSync	监控您的任务
Amazon DataZone		使用 Amazon CloudWatch 监控 Amazon DataZone
Amazon DevOps Guru	AWS/DevOps-Guru	使用 Amazon CloudWatch 监控 Amazon DevOps Guru
AWS Database Migration Service	AWS/DMS	监控 AWS DMS 任务
AWS Direct Connect	AWS/DX	使用 Amazon CloudWatch 进行监控
AWS Directory Service	AWS/DirectoryService	使用 Amazon CloudWatch 指标来确定何时添加域控制器
Amazon DocumentDB	AWS/DocDB	Amazon DocumentDB 指标
Amazon DynamoDB	AWS/DynamoDB	DynamoDB 指标与维度
DynamoDB Accelerator (DAX)	AWS/DAX	查看 DAX 指标和维度
Amazon EC2	AWS/EC2	使用 CloudWatch 监控您的实例

服务	命名空间	文档
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	使用 CloudWatch 指标监控 Elastic Graphics
Amazon EC2 Spot 实例集	AWS/EC2Spot	Spot 队列的 CloudWatch 指标
Amazon EC2 Auto Scaling	AWS/AutoScaling	使用 CloudWatch 监控您的 Auto Scaling 组和实例
AWS Elastic Beanstalk	AWS/ElasticBeanstalk	发布环境的 Amazon CloudWatch 自定义指标
Amazon Elastic Block Store	AWS/EBS	Amazon EBS 的 Amazon CloudWatch 指标
Amazon Elastic Container Registry	AWS/ECR	Amazon ECR 存储库指标
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch 指标
Amazon ECS - CloudWatch Container Insights	ECS/ContainerInsights	Amazon ECS Container Insights 指标
Amazon ECS 集群弹性伸缩	AWS/ECS/ManagedScaling	Amazon ECS 集群自动扩展
AWS Elastic Disaster Recovery		DRS 的 CloudWatch 指标

服务	命名空间	文档
Amazon Elastic File System	AWS/EFS	使用 CloudWatch 进行监控
Amazon Elastic Inference	AWS/ElasticInference	使用 CloudWatch 指标监测 Amazon Elastic Inference
Amazon EKS - CloudWatch Container Insights	ContainerInsights	Amazon EKS 和 Kubernetes Container Insights 指标
Elastic Load Balancing	AWS/ApplicationELB	Application Load Balancer 的 CloudWatch 指标
Elastic Load Balancing	AWS/NetworkELB	Network Load Balancer 的 CloudWatch 指标
Elastic Load Balancing	AWS/GatewayELB	网关负载均衡器的 CloudWatch 指标
Elastic Load Balancing	AWS/ELB	经典负载均衡器的 CloudWatch 指标
Amazon Elastic Transcoder	AWS/ElasticTranscoder	使用 Amazon CloudWatch 进行监控
Amazon ElastiCache (Memcached)	AWS/ElastiCache	使用 CloudWatch 指标监控使用情况
Amazon ElastiCache (Redis OSS)	AWS/ElastiCache	使用 CloudWatch 指标监控使用情况

服务	命名空间	文档
Amazon OpenSearch Service	AWS/ES	使用 Amazon CloudWatch 监控 OpenSearch 集群指标
Amazon EMR	AWS/ElasticMapReduce	使用 CloudWatch 监控指标
AWS Elemental MediaConnect	AWS/MediaConnect	使用 Amazon CloudWatch 监控 MediaConnect
AWS Elemental MediaConvert	AWS/MediaConvert	使用 CloudWatch 指标查看 AWS Elemental MediaConvert 资源的指标
AWS Elemental MediaLive	AWS/MediaLive	使用 Amazon CloudWatch 指标监控活动
AWS Elemental MediaPackage	AWS/MediaPackage	使用 Amazon CloudWatch 指标监控 AWS Elemental MediaPackage
AWS Elemental MediaStore	AWS/MediaStore	使用 Amazon CloudWatch 指标监控 AWS Elemental MediaStore
AWS Elemental MediaTailor	AWS/MediaTailor	使用 Amazon CloudWatch 监控 AWS Elemental MediaTailor
Amazon EventBridge	AWS/Events	监控 Amazon EventBridge
Amazon FinSpace		日志记录和监控
Amazon Forecast		Amazon Forecast 的 CloudWatch 指标
Amazon Fraud Detector		使用 Amazon CloudWatch 监控 Amazon Fraud Detector

服务	命名空间	文档
Amazon FSx for Lustre	AWS/FSx	监控 Amazon FSx for Lustre
Amazon FSx for OpenZFS	AWS/FSx	使用 Amazon CloudWatch 进行监控
适用于 Windows File Server 的 Amazon FSx	AWS/FSx	监控适用于 Windows File Server 的 Amazon FSx
Amazon FSx for NetApp ONTAP	AWS/FSx	使用 Amazon CloudWatch 进行监控
Amazon FSx for OpenZFS	AWS/FSx	使用 Amazon CloudWatch 进行监控
Amazon GameLift	AWS/GameLift	使用 CloudWatch 监控 Amazon GameLift
AWS Global Accelerator	AWS/GlobalAccelerator	使用 Amazon CloudWatch 与 AWS Global Accelerator
AWS Glue	Glue	使用 CloudWatch 指标监控 AWS Glue
AWS Ground Station	AWS/GroundStation	使用 Amazon CloudWatch 的指标
AWS HealthLake	AWS/HealthLake	使用 CloudWatch 监控 HealthLake
Amazon Inspector	AWS/Inspector	使用 CloudWatch 监控 Amazon Inspector
Amazon Interactive Video Service	AWS/IVS	使用 Amazon CloudWatch 监控 Amazon IVS

服务	命名空间	文档
Amazon Interactive Video Service Chat	AWS/IVSChat	使用 Amazon CloudWatch 监控 Amazon IVS
AWS IoT	AWS/IoT	AWS IoT 指标与维度
AWS IoT Analytics	AWS/IoTAnalytics	命名空间、指标和维度
AWS IoT FleetWise	AWS/IoTFleetWise	使用 Amazon CloudWatch 监控 AWS IoT FleetWise
AWS IoT SiteWise	AWS/IoTSiteWise	使用 Amazon CloudWatch 指标监控 AWS IoT SiteWise
AWS IoT TwinMaker	AWS/IoTTwinMaker	使用 Amazon CloudWatch 指标监控 AWS IoT TwinMaker
AWS IoT一键		使用 Amazon CloudWatch 监控 AWS IoT 1-Click
AWS Key Management Service	AWS/KMS	使用 CloudWatch 进行监控
Amazon Keyspaces（Apache Cassandra 兼容）	AWS/Cassandra	Amazon Keyspaces 指标与维度
Amazon Kendra		使用 Amazon CloudWatch 监控 Amazon Kendra
适用于 Apache Flink 的亚马逊托管服务	AWS/KinesisAnalytics	适用于 Apache Flink for SQL Applications 的托管服务：使用 CloudWatch 进行监控 适用于 Apache Flink 的 Apache Flink 托管服务：查看适用于 Apache Flink 的亚马逊托管服务指标和维度

服务	命名空间	文档
Amazon Data Firehose	AWS/Firehose	使用 CloudWatch 指标监控 Firehose
Amazon Kinesis Data Streams	AWS/Kinesis	使用 Amazon CloudWatch 监控 Amazon Kinesis Data Streams
Amazon Kinesis Video Streams	AWS/KinesisVideo	使用 CloudWatch 监控 Kinesis Video Streams 指标
AWS Lambda	AWS/Lambda	AWS Lambda 指标
Amazon Lex	AWS/Lex	使用 Amazon CloudWatch 监控 Amazon Lex
AWS License Manager	AWSLicenseManager/licenseUsage AWS/LicenseManager/LinuxSubscriptions	使用 Amazon CloudWatch 监控许可证使用情况 Linux 订阅的使用情况指标和 Amazon CloudWatch 警报
Amazon Location Service	AWS/Location	导出到 Amazon CloudWatch 的 Amazon Location Service 指标
Amazon Lookout for Equipment	AWS/lookoutequipment	使用 Amazon CloudWatch 监控 Lookout for Equipment
Amazon Lookout for Metrics	AWS/LookoutMetrics	使用 Amazon CloudWatch 监控 Lookout for Metrics
Amazon Lookout for Vision	AWS/LookoutVision	使用 Amazon CloudWatch 监控 Lookout for Vision

服务	命名空间	文档
AWS Mainframe Modernization		使用 Amazon CloudWatch 监控 AWS Mainframe Modernization
Amazon Machine Learning	AWS/ML	使用 CloudWatch 指标监控 Amazon ML
Amazon Managed Blockchain	AWS/managedblockchain	在 Amazon Managed Blockchain 上使用 Hyperledger Fabric 对等节点指标
Amazon Managed Service for Prometheus	AWS/Prometheus	Amazon CloudWatch 指标
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	使用 Amazon CloudWatch 监控 Amazon MSK
Amazon Managed Streaming for Apache Kafka	AWS/KafkaConnect	监控 MSK Connect
Amazon Managed Workflows for Apache Airflow	AWS/MWAA	Amazon MWAA 的容器、队列和数据库指标
Amazon MemoryDB	AWS/MemoryDB	监控 CloudWatch 指标
Amazon MQ	AWS/AmazonMQ	使用 Amazon CloudWatch 监控 Amazon MQ 代理

服务	命名空间	文档
Amazon Neptune	AWS/Neptune	使用 CloudWatch 监控 Neptune
AWS Network Firewall	AWS/NetworkFirewall	Amazon CloudWatch 中的 AWS Network Firewall 指标
AWS 网络管理器	AWS/NetworkManager	适用于本地资源的 CloudWatch 指标
Amazon Nimble Studio	AWS/NimbleStudio	使用 Amazon CloudWatch 监控 Nimble Studio
AWS HealthOmics	AWS/Omics	使用 Amazon CloudWatch 监控 AWS HealthOmics
AWS OpsWorks	AWS/OpsWorks	使用 Amazon CloudWatch 监控堆栈
AWS Outposts	AWS/Outposts	AWS Outposts 的 CloudWatch 指标
AWS Panorama	AWS/PanoramaDeviceMetrics	使用 Amazon CloudWatch 监控设备和应用程序
Amazon Personalize	AWS/Personalize	Amazon Personalize 的 CloudWatch 指标
Amazon Pinpoint	AWS/Pinpoint	查看 CloudWatch 中的 Amazon Pinpoint 指标
Amazon Polly	AWS/Polly	将 CloudWatch 与 Amazon Polly 集成
AWS PrivateLink	AWS/PrivateLinkEndpoints	AWS PrivateLink 的 CloudWatch 指标

服务	命名空间	文档
AWS PrivateLink	AWS/PrivateLinkServices	AWS PrivateLink 的 CloudWatch 指标
AWS 私有 5G	AWS/Private5G	Amazon CloudWatch 指标
Amazon QLDB	AWS/QLDB	监控 Amazon QuickSight 中的数据
Amazon QuickSight	AWS/QuickSight	使用 Amazon CloudWatch 进行监控
Amazon Redshift	AWS/Redshift	Amazon Redshift 性能数据
Amazon Relational Database Service	AWS/RDS	使用 Amazon CloudWatch 监控 Amazon RDS 指标
Amazon Rekognition	AWS/Rekognition	使用 Amazon CloudWatch 监控 Rekognition
AWS re:Post Private	AWS/rePostPrivate	使用 Amazon CloudWatch 监控 AWS re:Post Private
AWS RoboMaker	AWS/Robomaker	使用 Amazon CloudWatch 监控 AWS RoboMaker
Amazon Route 53	AWS/Route53	监控 Amazon Route 53
Route 53 应用程序恢复控制器	AWS/Route53RecoveryReadiness	将 Amazon CloudWatch 与应用程序恢复控制器配合使用

服务	命名空间	文档
Amazon SageMaker	AWS/SageMaker	使用 CloudWatch 监控 SageMaker
Amazon SageMaker 模型构建管道	AWS/SageMaker/ModelBuildingPipeline	SageMaker 管道指标
AWS Secrets Manager	AWS/SecretsManager	使用 Amazon CloudWatch 监控 Secrets Manager
Amazon Security Lake	AWS/SecurityLake	Amazon Security Lake 的 CloudWatch 指标
服务目录	AWS/ServiceCatalog	Service Catalog CloudWatch 指标
AWS Shield Advanced	AWS/DDoSProtection	使用 CloudWatch 进行监控
Amazon Simple Email Service	AWS/SES	从 CloudWatch 检索 Amazon SES 事件数据
AWS SimSpace Weaver	AWS/simspaceweaver	使用 Amazon CloudWatch 监控 AWS SimSpace Weaver
Amazon Simple Notification Service	AWS/SNS	使用 CloudWatch 监控 Amazon SNS
Amazon Simple Queue Service	AWS/SQS	使用 CloudWatch 监控 Amazon SQS 队列
Amazon S3	AWS/S3	使用 Amazon CloudWatch 监控指标

服务	命名空间	文档
S3 Storage Lens 存储统计管理工具	AWS/S3/Storage-Lens	在 CloudWatch 中监控 S3 Storage Lens 指标
Amazon Simple Workflow Service	AWS/SWF	用于 CloudWatch 的 Amazon SWF 指标
AWS Step Functions	AWS/States	使用 CloudWatch 监控 Step Functions
AWS Storage Gateway	AWS/StorageGateway	使用 Amazon CloudWatch 指标
AWS Systems Manager Run Command	AWS/SSM-RunCommand	使用 CloudWatch 监控 Run Command 指标
Amazon Textract	AWS/Textract	Amazon Textract 的 CloudWatch 指标
Amazon Timestream	AWS/Timestream	Timestream 指标与维度
AWS Transfer for SFTP	AWS/Transfer	AWS SFTP CloudWatch 指标
Amazon Transcribe	AWS/Transcribe	使用 Amazon CloudWatch 监控 Amazon Transcribe
Amazon Translate	AWS/Translate	Amazon Translate 的 CloudWatch 指标与维度
AWS Trusted Advisor	AWS/TrustedAdvisor	使用 CloudWatch 创建 Trusted Advisor 警报

服务	命名空间	文档
Amazon VPC	AWS/NATGateway	使用 CloudWatch 监控 NAT 网关
Amazon VPC	AWS/TransitGateway	Transit Gateway 的 CloudWatch 指标
Amazon VPC	AWS/VPN	使用 CloudWatch 进行监控
Amazon VPC IP 地址管理器	AWS/IPAM	使用 Amazon CloudWatch 创建告警
AWS WAF	用于 AWS WAF 资源的 AWS/WAFV2 用于 AWS WAF Classic 资源的 WAF	使用 CloudWatch 进行监控
Amazon WorkMail	AWS/WorkMail	使用 Amazon CloudWatch 监控 Amazon WorkMail
Amazon WorkSpaces	AWS/WorkSpaces	使用 CloudWatch 指标监控您的 WorkSpaces
Amazon WorkSpaces Web	AWS/WorkSpacesWeb	使用 Amazon CloudWatch 监控 Amazon WorkSpaces Web

AWS 使用情况指标

CloudWatch 会收集跟踪某些 AWS 资源和 API 的使用情况的指标。这些指标发布在 AWS/Usage 命名空间中。CloudWatch 中的使用情况指标允许您通过在 CloudWatch 控制台中直观呈现指标、创建自定义控制面板、使用 CloudWatch 异常检测功能检测活动变化以及配置在使用量接近阈值时提示您的告警，主动管理使用情况。

一些 AWS 服务将这些使用情况指标与 Service Quotas 集成。对于这些服务，您可以使用 CloudWatch 管理您的账户对 Service Quotas 的使用情况。有关更多信息，请参阅 可视化 Service Quotas 并设置告警。

主题

- 可视化 Service Quotas 并设置告警
- AWS API 使用情况指标
- CloudWatch 使用情况指标

可视化 Service Quotas 并设置告警

对于一些 AWS 服务，您可以使用使用情况指标在 CloudWatch 图表和控制面板上直观呈现当前服务使用情况。您可以使用 CloudWatch 指标数学函数在图表上显示这些资源的服务配额。还可以配置警报，以在用量接近服务配额时向您发出警报。有关 Service Quotas 的更多信息，请参阅 Service Quotas 用户指南中的什么是 Service Quotas。

如果您登录的账户在 CloudWatch 跨账户可观测性中设置为监控账户，则您可以使用该监控账户来可视化服务限额，并为链接到该监控账户的源账户中的指标设置警报。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

目前，以下服务将其使用情况指标与 Service Quotas 集成：

- AWS CloudHSM
- Amazon Chime SDK
- Amazon CloudWatch
- Amazon CloudWatch Logs
- Amazon DynamoDB
- Amazon EC2
- Amazon Elastic Container Registry

- Elastic Load Balancing
- AWS Fargate
- AWS Fault Injection Service
- AWS Interactive Video Service
- AWS Key Management Service
- Amazon Data Firehose
- Amazon Location Service
- Amazon Managed Blockchain（AMB）查询
- AWS Robomaker
- Amazon SageMaker

可视化服务配额并选择性地设置警报

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 在 All metrics（所有指标）选项卡上，选择 Usage（使用情况），然后选择 By AWS Resource（按亚马逊云科技资源）。

   这将显示服务配额用量指标的列表。
4. 选中其中一个指标旁边的复选框。

   该图表显示您该 AWS 资源的当前用量。
5. 要将服务配额添加到图表，请执行以下操作：

   a. 选择 Graphed metrics（绘制的指标）选项卡。

   b. 选择数学表达式、从空表达式开始。在新行中，在 Details（详细信息）下，输入 **`SERVICE_QUOTA(m1)`**。

      这将向图表中添加一个新行，并显示指标中表示的资源的服务配额。
6. 要以配额百分比的形式查看您的当前用量，请添加新的表达式或更改当前 SERVICE_QUOTA 表达式。使用的新表达式是 **`m1/SERVICE_QUOTA(m1)*100`**。
7. (可选) 要设置一个警报，以便在接近服务配额时向您发送通知，请执行以下操作：

   a. 在 **`m1/SERVICE_QUOTA(m1)*100`** 行上的 Action（操作）下，选择告警图标。该图标看起来像一个铃铛。

这将显示警报创建页面。

b. 在条件下，确保阈值类型为静态，并将当 Expression1 为设置为大于。在多于下，输入 **80**。这将创建一个警报，当用量超过配额的 80% 时，该警报将进入 ALARM 状态。

c. 选择下一步。

d. 在下一页上，选择 Amazon SNS 主题或创建一个新主题，然后选择 Next（下一步）。当告警进入 ALARM 状态时，会向此主题发送通知。

e. 在下一页上，输入警报的名称和描述，然后选择下一步。

f. 选择创建警报。

AWS API 使用情况指标

大多数支持 AWS CloudTrail 日志记录的 API 还会向 CloudWatch 报告使用情况指标。CloudWatch 中的 API 使用情况指标允许您通过在 CloudWatch 控制台中直观呈现指标、创建自定义控制面板、使用 CloudWatch 异常检测功能检测活动变化以及配置在使用量接近阈值时提示您的告警，主动管理 API 使用情况。

下表列出了向 CloudWatch 报告 API 使用情况指标的服务，以及 `Service` 维度用于查看该服务的使用情况指标的值。

服务	**`Service`** 维度的值
AWS Identity and Access Management Access Analyzer	`Access Analyzer`
AWS Account Management	`Account Management`
Alexa for Business	`A4B`
Amazon API Gateway	`API Gateway`
AWS App Mesh	`App Mesh`
AWS AppConfig	`AWS AppConfig`
Amazon AppFlow	`AppFlow`

服务	**Service** 维度的值
Application Auto Scaling	`Application Auto Scaling`
Application Discovery Service	`Application Discovery Service`
Amazon AppStream	`AppStream`
AppStream 2.0 Image Builder	`Image Builder`
Amazon Athena	`Athena`
AWS Audit Manager	`Audit Manager`
AWS Backup	`Backup`
AWS Batch	`Batch`
Amazon Braket	`Braket`
AWS Budgets	`Budgets`
AWS Certificate Manager	`Certificate Manager`
Amazon Chime SDK	`ChimeSDK`
Amazon Cloud Directory	`Cloud Directory`
AWS Cloud Map	`Cloud Map`
AWS CloudFormation	`CloudFormation`
AWS CloudHSM	`CloudHSM`
Amazon CloudSearch	`CloudSearch`
AWS CloudShell	`CloudShell`
AWS CloudTrail	`CloudTrail`
Amazon CloudWatch	`CloudWatch`

服务	**Service** 维度的值
Amazon CloudWatch Application Signals	`CloudWatch Application Signals`
Amazon CloudWatch Logs	`Logs`
Amazon CloudWatch Application Insights	`CloudWatch Application Insights`
AWS CodeBuild	`CodeBuild`
AWS CodeCommit	`CodeCommit`
Amazon CodeGuru Profiler	`CodeGuru Profiler`
AWS CodePipeline	`CodePipeline`
AWS CodeStar	`CodeStar`
AWS CodeStar 通知	`CodeStar Notifications`
AWS CodeStar 连接	`CodeStar Connections`
Amazon Cognito 身份池	`Cognito Identity Pools`
Amazon Cognito Sync	`Cognito Sync`
Amazon Comprehend	`Comprehend`
Amazon Comprehend Medical	`Comprehend Medical`
AWS Compute Optimizer	`ComputeOptimzier`
Amazon Connect	`Connect`
Amazon Connect Customer Profiles	`Customer Profiles`
AWS 成本和使用情况报告	`Cost and Usage Report`
AWS Cost Explorer	`Cost Explorer`
AWS Data Exchange	`Data Exchange`

服务	**Service** 维度的值
AWS Data Lifecycle Manager	`Data Lifecycle Manager`
AWS Database Migration Service	`Database Migration Service`
AWS DataSync	`DataSync`
AWS DeepLens	`AWS DeepLens`
Amazon Detective	`Detective`
Device Advisor	`Device Advisor`
AWS Direct Connect	`Direct Connect`
AWS Directory Service	`Directory Service`
DynamoDB Accelerator	`DynamoDBAccelerator`
Amazon EC2	`EC2`
EC2 Auto Scaling	`EC2 Auto Scaling`
Amazon Elastic Container Registry	`ECR Public`
Amazon Elastic Container Service	`ECS`
Amazon Elastic File System	`EFS`
Amazon Elastic Kubernetes Service	`EKS`
AWS Elastic Beanstalk	`Elastic Beanstalk`
Amazon Elastic Inference	`Elastic Inference`
Elastic Load Balancing	`Elastic Load Balancing`
Amazon EMR	`EMR Containers`
AWS Firewall Manager	`Firewall Manager`

服务	**Service** 维度的值
Amazon FSx	FSx
Amazon GameLift	GameLift
AWS Glue DataBrew	DataBrew
Amazon Managed Grafana	Grafana
AWS IoT Greengrass	Greengrass
AWS Ground Station	Ground Station
AWS Health API 和通知	AWS Health APIs And Notifications
Amazon Interactive Video Service	IVS
AWS IoT Core	IoT
AWS IoT一键	IoT 1-Click
AWS IoT Events	IoT Events
AWS IoT RoboRunner	IoT RoboRunner
AWS IoT SiteWise	IoT Sitewise
AWS IoT Wireless	IoT Wireless
Amazon Kendra	Kendra
Amazon Keyspaces (for Apache Cassandra)	Keyspaces
适用于 Apache Flink 的亚马逊托管服务	Kinesis Analytics
Amazon Data Firehose	Firehose
Kinesis Video Streams	Kinesis Video Streams
AWS Key Management Service	KMS

服务	**Service** 维度的值
AWS Lambda	`Lambda`
AWS Launch Wizard	`Launch Wizard`
Amazon Lex	`Amazon Lex`
Amazon Lightsail	`Lightsail`
Amazon Location Service	`Location`
Amazon Lookout for Vision	`Lookout for Vision`
Amazon Machine Learning	`Amazon Machine Learning`
Amazon Macie	`Macie`
Amazon Managed Blockchain（AMB）查询	`Amazon Managed Blockchain Query`
AWS Managed Services	`AWS Managed Services`
AWS Marketplace Commerce Analytics	`Marketplace Analytics Service`
AWS Elemental MediaConnect	`MediaConnect`
AWS Elemental MediaConvert	`MediaConvert`
AWS Elemental MediaLive	`MediaLive`
AWS Elemental MediaStore	`Mediastore`
AWS Elemental MediaTailor	`MediaTailor`
AWS Mobile Hub	`Mobile Hub`
AWS Network Firewall	`Network Firewall`
AWS OpsWorks	`OpsWorks`
用于配置管理的 AWS OpsWorks	`OPsWorks CM`

服务	**Service** 维度的值
AWS Outposts	`Outposts`
AWS Organizations	`Organizations`
Amazon RDS 性能详情	`Performance Insights`
Amazon Pinpoint	`Pinpoint`
AWS Private Certificate Authority	`Private Certificate Authority`
Amazon Managed Service for Prometheus	`Prometheus`
AWS Proton	`Proton`
Amazon Quantum Ledger Database (Amazon QLDB)	`QLDB`
Amazon RDS	`RDS`
Amazon Redshift	`Redshift Data API`
Amazon Rekognition	`Rekognition`
AWS Resource Access Manager	`Resource Access Manager`
AWS Resource Groups	`Resource Groups`
AWS Resource Groups Tagging API	`Resource Groups Tagging API`
AWS RoboMaker	`RoboMaker`
Amazon Route 53 Domains	`Route 53 Domains`
Amazon Route 53 Resolver	`Route 53 Resolver`
Amazon S3	`S3`
Amazon S3 Glacier	`Amazon S3 Glacier`
Amazon SageMaker Runtime	`Sagemaker`

服务	**Service** 维度的值
节省计划	`Savings Plans`
AWS Secrets Manager	`Secrets Manager`
AWS Security Hub	`Security Hub`
AWS Server Migration Service	`AWS Server Migration Service`
AWS Service Catalog AppRegistry	`Service Catalog AppRegistry`
Service Quotas	`Service Quotas`
AWS Shield	`Shield`
AWS Signer	`Signer`
Amazon Simple Notification Service	`SNS`
Amazon Simple Email Service	`SES`
Amazon Simple Queue Service	`SQS`
Identity Store	`Identity Store`
Storage Gateway	`Storage Gateway`
AWS Support	`Support`
Amazon Simple Workflow Service	`SWF`
Amazon Textract	`Textract`
AWS IoT Things Graph	`ThingsGraph`
Amazon Timestream	`Timestream`
Amazon Transcribe	`Transcribe`
Amazon Translate	`Translate`

服务	**Service** 维度的值
Amazon Transcribe 串流转录	`Transcribe Streaming`
AWS Transfer Family	`Transfer`
AWS WAF	`WAF`
Amazon WorkDocs	`Amazon WorkDocs`
Amazon WorkLink	`WorkLink`
Amazon WorkMail	`Amazon WorkMail`
Amazon WorkSpaces	`Workspaces`
AWS X-Ray	`X-Ray`

某些服务还会报告其他 API 的使用情况指标。若要查看 API 是否向 CloudWatch 报告使用情况指标，请使用 CloudWatch 控制台查看该服务在 `AWS/Usage` 命名空间中报告的指标。

查看向 CloudWatch 报告使用情况指标的服务 API 列表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在导航窗格中，选择指标。
3. 在 All metrics（所有指标）选项卡上，选择 Usage（使用情况），然后选择 By AWS Resource（按亚马逊云科技资源）。
4. 在指标列表旁边的搜索框中，输入服务的名称。指标将根据您输入的服务筛选出来。

CloudWatch 使用情况指标

CloudWatch 会收集跟踪某些 AWS 资源的使用情况的指标。这些指标与 AWS 服务配额对应。跟踪这些指标可帮助您主动管理配额。有关更多信息，请参阅可视化 Service Quotas 并设置告警。

服务配额使用指标位于 `AWS/Usage` 命名空间中，并且每分钟收集一次。

目前，CloudWatch 在此命名空间中发布的唯一指标名称是 `CallCount`。此指标与 `Resource`、`Service` 和 `Type` 维度一同发布。`Resource` 维度指定要跟踪的 API 操作的名称。例

如，具有 "Service": "CloudWatch"、"Type": "API" 和 "Resource": "PutMetricData" 维度的 CallCount 指标指示了 CloudWatch PutMetricData API 操作在您的账户中被调用的次数。

CallCount 指标没有指定的单位。该指标中最实用的统计数据是 SUM，它表示以 1 分钟为间隔的总操作数。

指标

指标	描述
CallCount	在您的账户中执行的指定操作的数量。

维度

维度	描述
Service	包含该资源的 AWS 服务的名称。对于 CloudWatch 使用情况指标，此维度的值为 CloudWatch 。
Class	所跟踪的资源的类。CloudWatch API 使用情况指标使用此维度（值为 None）。
Type	所跟踪的资源的类型。目前，当 Service 维度为 CloudWatch 时，Type 的唯一有效值为 API。
Resource	API 操作的名称。有效值包括：DeleteAlarms 、DeleteDashboards 、DescribeAlarmHistory 、DescribeAlarms 、GetDashboard 、GetMetricData 、GetMetricStatistics 、ListMetrics 、PutDashboard 和 PutMetricData

CloudWatch 教程

下面的方案说明如何使用 Amazon CloudWatch。在第一个方案中，您将使用 CloudWatch 控制台创建账单告警，以便跟踪您的 AWS 使用率并在超过特定消费阈值时通知您。在第二个更高级的方案中，您使用 AWS Command Line Interface (AWS CLI) 为名为 GetStarted 的假想应用程序发布单个指标。

场景

- 监控您的预估费用
- 发布指标

方案：使用 CloudWatch 监控您的估算费用

在此方案中，将创建一个 Amazon CloudWatch 告警来监控估计费用。在为您的 AWS 账户启用估计费用监控时，将每天计算几次估计费用并作为指标数据发送到 CloudWatch。

账单指标数据存储在美国东部（弗吉尼亚北部）区域中，并且反映全球费用。此数据包括您所用的每种 AWS 服务的估计费用以及 AWS 估计费用总额。

可以选择在收费超过特定阈值时通过电子邮件接收警报。这些告警由 CloudWatch 触发，消息使用 Amazon Simple Notification Service (Amazon SNS) 进行发送。

> **Note**
>
> 有关分析已计费的 CloudWatch 费用的信息，请参阅 CloudWatch 账单和成本。

任务

- 步骤 1：启用账单提醒
- 步骤 2：创建账单告警
- 步骤 3：检查告警状态
- 步骤 4：编辑账单告警
- 步骤 5：删除账单告警

步骤 1：启用账单提醒

在为估算费用创建告警之前，您必须启用账单提醒，以便监控 AWS 估算费用并使用账单指标数据创建告警。启用账单提醒后，您将无法禁用数据收集，但是可以删除任何已创建的账单警报。

首次启用账单提醒后，大约需要 15 分钟时间，您就可以查看账单数据和设置账单告警。

要求

- 您必须使用根用户凭证或作为被授予权限的用户登录，才能查看账单信息。
- 对于整合账单账户，每个关联账户的账单数据可以在付款账户登录后找到。您可以查看每个关联账户以及整合账户的估计费用总和，和各项服务的估计费用。
- 在整合账单账户中，仅当付款人账户启用 Receive Billing Alerts（接收账单提醒）首选项时，才会捕获成员关联账户的指标。如果您更改了您的管理账户/付款人账户，则必须在新的管理账户/付款人账户中启用账单提醒。
- 该账户不能属于 Amazon 合作伙伴网络 (APN)，因为对于 APN 账户，账单指标不会发布到 CloudWatch。有关更多信息，请参阅 AWS 合作伙伴网络。

要启用预估收费监控

1. 打开 AWS Billing 控制台，地址：https://console.aws.amazon.com/billing/。
2. 在导航窗格中，选择 Billing Preferences（账单首选项）。
3. 通过提醒首选项选择编辑。
4. 选择接收 CloudWatch 账单提醒。
5. 选择 Save preferences（保存首选项）。

步骤 2：创建账单告警

> **Important**
>
> 创建账单告警之前，您必须将区域设置为美国东部（弗吉尼亚州北部）。账单指标数据存储在该区域中，并表示全球费用。您还必须为您的账户启用账单提醒；或者，如果您使用的是整合账单，则必须在管理账户/付款人账户中启用账单提醒。有关更多信息，请参阅步骤 1：启用账单提醒。

在该过程中，您可以创建一个告警，以便在 AWS 的估算费用超出定义的阈值时发送通知。

使用 CloudWatch 控制台创建告警

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择 Alarms（告警），然后选择 All alarms（所有告警）。
3. 选择Create alarm（创建警报）。
4. 选择选择指标。在 Browse（浏览）中，选择 Billing（账单），然后选择 Total Estimated Charge（总估算费用）。

   > **Note**
   >
   > 如果您没看到账单/总估算费用指标，则启用账单提醒，并将您的区域更改为美国东部（弗吉尼亚州北部）。有关更多信息，请参阅启用账单提醒。

5. 选中 EstimatedCharges（估算费用）指标的复选框，然后选择 Select metric（选择指标）。
6. 对于 Statistic（统计数据），选择 Maximum（最大）。
7. 对于 Period（周期），选择 6 hours（6 小时）。
8. 对于 Threshold type（阈值类型），选择 Static（静态）。
9. 对于 Whenever EstimatedCharges is . . .（当估算费用. .），选择 Greater（大）。
10. 对于 than . . .，请定义要触发告警的值。例如，**200** USD。

    EstimatedCharges 指标值仅以美元（USD）为单位，货币转换由 Amazon Services LLC 提供。有关更多信息，请参阅什么是 AWS Billing？。

    > **Note**
    >
    > 定义阈值后，预览图显示您当月的预估费用。

11. 在其他配置中，执行以下操作：
    - 对于 Datapoints to alarm（触发告警的数据点数），指定 1 out of 1（1 选 1）。
    - 对于 Missing data treatment（缺失数据处理），选择 Treat missing data as missing（将缺失的数据视为缺失）。
12. 选择 Next（下一步）。

13. 在通知下，确保选择告警中。选择当您的告警处于 ALARM 状态时要通知的 Amazon SNS 主题。Amazon SNS 主题可以包含您的电子邮件地址，这样当账单金额超过您指定的阈值时，您就可以收到电子邮件。

    您可以选择现有的 Amazon SNS 主题、创建一个新 Amazon SNS 主题或使用主题 ARN 通知其他账户。如果您希望您的告警为相同告警状态或不同告警状态发送多个通知，请选择 Add notification（添加通知）。

14. 选择 Next（下一步）。
15. 在 Name and description（名称和描述）下，为您的告警输入名称。

    - （可选）输入告警的描述。

16. 选择 Next（下一步）。
17. 在 Preview and create（预览和创建）下，确保您的配置正确，然后选择 Create alarm（创建告警）。

步骤 3：检查告警状态

现在，请检查您刚创建的账单警报的状态。

检查警报状态

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 如果需要，可将区域更改为美国东部（弗吉尼亚北部）。账单指标数据存储在此区域中，并且反映全球费用。
3. 在导航窗格中，选择 Alarms (告警)。
4. 选中警报旁边的复选框。在确认订阅之前，它显示为“等待确认”。在确认订阅后，请刷新控制台以显示更新后的状态。

步骤 4：编辑账单告警

例如，您可能想要将每个月在 AWS 上花费的金额从 200 美元增加至 400 美元。您可以编辑现有账单警报并增加在触发警报前必须超过的金额。

编辑账单警报

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。

2. 如果需要，可将区域更改为美国东部（弗吉尼亚北部）。账单指标数据存储在此区域中，并且反映全球费用。
3. 在导航窗格中，选择 Alarms (告警)。
4. 选中警报旁的复选框，然后依次选择 Actions (操作) 和 Modify (修改)。
5. 对于 Whenever my total AWS charges for the month exceed（每当该月我的亚马逊云科技费用总额超过时），请指定必须超过才会触发告警并发送电子邮件通知的新金额。
6. 选择 Save Changes。

步骤 5：删除账单告警

如果您不再需要账单警报，可将其删除。

删除账单告警

1. 通过以下网址打开 CloudWatch 控制台：https://console.aws.amazon.com/cloudwatch/。
2. 如果需要，可将区域更改为美国东部（弗吉尼亚北部）。账单指标数据存储在此区域中，并且反映全球费用。
3. 在导航窗格中，选择 Alarms (告警)。
4. 选中告警旁的复选框，然后依次选择 Actions（操作） 和 Delete（删除）。
5. 当系统提示进行确认时，选择 Yes, Delete（是，删除）。

方案：将指标发布到 CloudWatch

在此方案中，使用 AWS Command Line Interface (AWS CLI) 为名为 GetStarted 的假想应用程序发布单个指标。如果您尚未安装和配置 AWS CLI，请参阅 AWS Command Line Interface 用户指南中的开始设置 AWS Command Line Interface。

任务

- 步骤 1：定义数据配置
- 步骤 2：将指标添加到 CloudWatch
- 步骤 3：获取 CloudWatch 中的统计数据
- 步骤 4：使用控制台查看图表

步骤 1：定义数据配置

在此场景中，您将发布跟踪该应用程序的请求延迟的数据点。为您的指标和命名空间选择您能够理解的名称。对本例而言，可以将指标命名为 RequestLatency 并将所有的数据点发布到 GetStarted 命名空间。

您将会发布共表示三小时延迟数据的几个数据点。原始数据由分布在三小时中的 15 个请求延迟读数构成。每个读数均以毫秒为单位：

- 第一小时：87、51、125、235
- 第二小时：121、113、189、65、89
- 第三小时：100、47、133、98、100、328

您可以将数据作为单一数据点或数据点聚合集发布到 CloudWatch，聚合集称为统计数据集。
您可以在低达一分钟的时间内将指标整合到粒度级。您可以将聚合数据点作为含四个预定义键（Sum、Minimum、Maximum 和 SampleCount）的统计数据集发布到 CloudWatch。

您将会把来自第一小时的数据点作为单一数据点发布。对于第二小时和第三小时的数据，您将会整合数据点并发布各小时的统计数据集。项值如下表所示。

小时	原始数据	总计	最低	最高	样本数
1	87				
1	51				
1	125				
1	235				
2	121, 113, 189, 65, 89	577	65	189	5
3	100, 47, 133, 98, 100, 328	806	47	328	6

步骤 2：将指标添加到 CloudWatch

定义数据配置后，您可以随时添加数据。

将数据点发布到 CloudWatch

1. 在命令提示符下，运行以下 put-metric-data 命令以添加第一个小时的数据。将示例时间戳替换为通用协调时间 (UTC) 中过去两小时的时间戳。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 87 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 51 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 125 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T20:30:00Z --value 235 --unit Milliseconds
```

2. 添加第二个小时的数据，并使用比第一个小时晚一小时的时间戳。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--timestamp 2016-10-14T21:30:00Z --statistic-values
 Sum=577,Minimum=65,Maximum=189,SampleCount=5 --unit Milliseconds
```

3. 添加第三个小时的数据，并忽略默认为当前时间的时间戳。

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted
 \
--statistic-values Sum=806,Minimum=47,Maximum=328,SampleCount=6 --unit Milliseconds
```

步骤 3：获取 CloudWatch 中的统计数据

您现在已向 CloudWatch 发布了指标，可以使用 get-metric-statistics 命令检索基于这些指标的统计数据（如下所示）。确保指定足够久的过去 `--start-time` 和 `--end-time` 以涵盖已发布的最早时间戳。

```
aws cloudwatch get-metric-statistics --namespace GetStarted --metric-name
 RequestLatency --statistics Average \
--start-time 2016-10-14T00:00:00Z --end-time 2016-10-15T00:00:00Z --period 60
```

下面是示例输出：

```
{
 "Datapoints": [],
 "Label": "Request:Latency"
}
```

步骤 4：使用控制台查看图表

将指标发布到 CloudWatch 后，您可以使用 CloudWatch 控制台查看统计图表。

如需在控制台中查看统计数据图表

1. 访问 https://console.aws.amazon.com/cloudwatch/ 打开 CloudWatch 控制台。
2. 在 Navigation 窗格中，选择 Metrics。
3. 在全部指标选项卡的搜索框中，键入 RequestLatency 并按 Enter。
4. 选中 RequestLatency 指标的复选框。上方窗格中会显示一个指标数据图表。

有关更多信息，请参阅绘制指标的图表。

将 CloudWatch 与 AWS SDK 结合使用

AWS 软件开发工具包 (SDK) 适用于许多常用编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例
AWS Tools for PowerShell	Tools for PowerShell 代码示例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

有关特定于 CloudWatch 的示例，请参阅 使用 AWS SDK 的 CloudWatch 代码示例。

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

使用 AWS SDK 的 CloudWatch 代码示例

以下代码示例展示如何将 CloudWatch 与 AWS 软件开发工具包 (SDK) 结合使用。

基础知识是向您展示如何在服务中执行基本操作的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务 结合来完成特定任务的代码示例。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

开始使用 CloudWatch

以下代码示例显示如何开始使用 CloudWatch。

.NET

AWS SDK for .NET

> **Note**
>
> 在 GitHub 上查看更多内容。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
```

```
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 ListMetrics。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 ListMetrics。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 ListMetrics。

代码示例

- 使用 AWS SDK 的 CloudWatch 基础知识示例
  - 开始使用 CloudWatch
  - 了解将 CloudWatch 与 AWS SDK 结合使用的基础知识
  - 使用 AWS SDK 的 CloudWatch 操作
    - 将 DeleteAlarms 与 AWS SDK 或 CLI 配合使用
    - 将 DeleteAnomalyDetector 与 AWS SDK 或 CLI 配合使用
    - 将 DeleteDashboards 与 AWS SDK 或 CLI 配合使用
    - 将 DescribeAlarmHistory 与 AWS SDK 或 CLI 配合使用
    - 将 DescribeAlarms 与 AWS SDK 或 CLI 配合使用
    - 将 DescribeAlarmsForMetric 与 AWS SDK 或 CLI 配合使用
    - 将 DescribeAnomalyDetectors 与 AWS SDK 或 CLI 配合使用
    - 将 DisableAlarmActions 与 AWS SDK 或 CLI 配合使用
    - 将 EnableAlarmActions 与 AWS SDK 或 CLI 配合使用
    - 将 GetDashboard 与 AWS SDK 或 CLI 配合使用
    - 将 GetMetricData 与 AWS SDK 或 CLI 配合使用
    - 将 GetMetricStatistics 与 AWS SDK 或 CLI 配合使用
    - 将 GetMetricWidgetImage 与 AWS SDK 或 CLI 配合使用
    - 将 ListDashboards 与 AWS SDK 或 CLI 配合使用
    - 将 ListMetrics 与 AWS SDK 或 CLI 配合使用
    - 将 PutAnomalyDetector 与 AWS SDK 或 CLI 配合使用
    - 将 PutDashboard 与 AWS SDK 或 CLI 配合使用
    - 将 PutMetricAlarm 与 AWS SDK 或 CLI 配合使用
    - 将 PutMetricData 与 AWS SDK 或 CLI 配合使用
- 使用 AWS SDK 的 CloudWatch 场景
  - 通过 AWS SDK 开始使用 CloudWatch 告警
  - 使用 AWS SDK 管理 CloudWatch 指标和告警
  - 使用 AWS SDK 监控 Amazon DynamoDB 的性能

使用 AWS SDK 的 CloudWatch 基础知识示例

以下代码示例展示了如何将 Amazon CloudWatch 的基本功能与 AWS SDK 结合使用。

示例

- 开始使用 CloudWatch
- 了解将 CloudWatch 与 AWS SDK 结合使用的基础知识
- 使用 AWS SDK 的 CloudWatch 操作
  - 将 DeleteAlarms 与 AWS SDK 或 CLI 配合使用
  - 将 DeleteAnomalyDetector 与 AWS SDK 或 CLI 配合使用
  - 将 DeleteDashboards 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeAlarmHistory 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeAlarms 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeAlarmsForMetric 与 AWS SDK 或 CLI 配合使用
  - 将 DescribeAnomalyDetectors 与 AWS SDK 或 CLI 配合使用
  - 将 DisableAlarmActions 与 AWS SDK 或 CLI 配合使用
  - 将 EnableAlarmActions 与 AWS SDK 或 CLI 配合使用
  - 将 GetDashboard 与 AWS SDK 或 CLI 配合使用
  - 将 GetMetricData 与 AWS SDK 或 CLI 配合使用
  - 将 GetMetricStatistics 与 AWS SDK 或 CLI 配合使用
  - 将 GetMetricWidgetImage 与 AWS SDK 或 CLI 配合使用
  - 将 ListDashboards 与 AWS SDK 或 CLI 配合使用
  - 将 ListMetrics 与 AWS SDK 或 CLI 配合使用
  - 将 PutAnomalyDetector 与 AWS SDK 或 CLI 配合使用
  - 将 PutDashboard 与 AWS SDK 或 CLI 配合使用
  - 将 PutMetricAlarm 与 AWS SDK 或 CLI 配合使用
  - 将 PutMetricData 与 AWS SDK 或 CLI 配合使用

开始使用 CloudWatch

以下代码示例显示如何开始使用 CloudWatch。

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
```

```
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"\tMetric: {metric.MetricName}");
            Console.WriteLine($"\tNamespace: {metric.Namespace}");
            Console.WriteLine($"\tDimensions: {string.Join(", ",
 metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 ListMetrics。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
```

```
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

```
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 ListMetrics。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
```

```
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 ListMetrics。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

了解将 CloudWatch 与 AWS SDK 结合使用的基础知识

以下代码示例演示了如何：

- 列出 CloudWatch 命名空间和指标。
- 获取指标和预估账单的统计数据。
- 创建和更新控制面板。
- 创建数据并将数据添加到指标。
- 创建并触发告警，然后查看告警历史记录。
- 添加异常检测器
- 获取指标映像，然后清理资源。

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

在命令提示符中运行交互式场景。

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */
```

```
   private static ILogger logger = null!;
   private static CloudWatchWrapper _cloudWatchWrapper = null!;
   private static IConfiguration _configuration = null!;
   private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
   private static SingleMetricAnomalyDetector? anomalyDetector = null!;

   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
           services.AddAWSService<IAmazonCloudWatch>()
           .AddTransient<CloudWatchWrapper>()
       )
       .Build();

       _configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
           .AddJsonFile("settings.json") // Load settings from .json file.
           .AddJsonFile("settings.local.json",
               true) // Optionally, load local settings.
           .Build();

       logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
           .CreateLogger<CloudWatchScenario>();

       _cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();

       Console.WriteLine(new string('-', 80));
       Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
       Console.WriteLine(new string('-', 80));

       try
       {
           var selectedNamespace = await SelectNamespace();
```

```
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace,
selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }
```

```
        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d
=> d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
```

```
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string
metricNamespace, Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last
day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the
preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };
```

```
        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for
{metric} in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
```

```
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
```

```
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
{customMetricName}: \n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
```

```
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
dashboardName)
    {
        // Get the dashboard model from configuration.
```

```
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm
{alarmName}? (y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");
```

```
        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for
{alarm.MetricName} {alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
```

```
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            for (int j = 0; j < metricData[i].Values.Count; j++)
            {
                Console.WriteLine(
                    $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger
an alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
```

```
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric
{customMetricName}: \n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
```

```
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10
retries.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state
{detector.StateValue}");
```

```
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream,
"MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
```

```
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
 anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
```

```
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

场景用于 CloudWatch 操作的包装程序方法。

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
 listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
 DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
```

```
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });
```

```
        return metricStatistics.Datapoints;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }
```

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
```

```
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

```
   /// <summary>
   /// Add a metric alarm to send an email when the metric passes a threshold.
   /// </summary>
   /// <param name="alarmDescription">A description of the alarm.</param>
   /// <param name="alarmName">The name for the alarm.</param>
   /// <param name="comparison">The type of comparison to use.</param>
   /// <param name="metricName">The name of the metric for the alarm.</param>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="threshold">The threshold value for the alarm.</param>
   /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
       string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
   {
       try
       {
           var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
               new PutMetricAlarmRequest()
               {
                   AlarmActions = alarmActions,
                   AlarmDescription = alarmDescription,
                   AlarmName = alarmName,
                   ComparisonOperator = comparison,
                   Threshold = threshold,
                   Namespace = metricNamespace,
                   MetricName = metricName,
                   EvaluationPeriods = 1,
                   Period = 10,
                   Statistic = new Statistic("Maximum"),
                   DatapointsToAlarm = 1,
                   TreatMissingData = "ignore"
               });
           return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (LimitExceededException lex)
       {
           _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
       }
```

```
        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
 alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
 topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
 append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
 null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
 _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
```

```
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
```

```
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
```

```
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
```

```
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });
```

```
        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的以下主题。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonParser;
```

```
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.AlarmHistoryItem;
import software.amazon.awssdk.services.cloudwatch.model.AlarmType;
import software.amazon.awssdk.services.cloudwatch.model.AnomalyDetector;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ComparisonOperator;
import
 software.amazon.awssdk.services.cloudwatch.model.DashboardValidationMessage;
import software.amazon.awssdk.services.cloudwatch.model.Datapoint;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DeleteAnomalyDetectorRequest;
import software.amazon.awssdk.services.cloudwatch.model.DeleteDashboardsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricResponse;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsResponse;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageResponse;
import software.amazon.awssdk.services.cloudwatch.model.HistoryItemType;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
```

```
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;
import software.amazon.awssdk.services.cloudwatch.model.MetricAlarm;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataQuery;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataResult;
import software.amazon.awssdk.services.cloudwatch.model.MetricDatum;
import software.amazon.awssdk.services.cloudwatch.model.MetricStat;
import
 software.amazon.awssdk.services.cloudwatch.model.PutAnomalyDetectorRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardResponse;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.ScanBy;
import
 software.amazon.awssdk.services.cloudwatch.model.SingleMetricAnomalyDetector;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.Statistic;
import
 software.amazon.awssdk.services.cloudwatch.paginators.ListDashboardsIterable;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.Instant;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To enable billing metrics and statistics for this example, make sure billing
 * alerts are enabled for your account:
 * https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics
 *
 * This Java code example performs the following tasks:
 *
 * 1. List available namespaces from Amazon CloudWatch.
 * 2. List available metrics within the selected Namespace.
 * 3. Get statistics for the selected metric over the last day.
 * 4. Get CloudWatch estimated billing for the last week.
 * 5. Create a new CloudWatch dashboard with metrics.
 * 6. List dashboards using a paginator.
 * 7. Create a new custom metric by adding data for it.
 * 8. Add the custom metric to the dashboard.
 * 9. Create an alarm for the custom metric.
 * 10. Describe current alarms.
 * 11. Get current data for the new custom metric.
 * 12. Push data into the custom metric to trigger the alarm.
 * 13. Check the alarm state using the action DescribeAlarmsForMetric.
 * 14. Get alarm history for the new alarm.
 * 15. Add an anomaly detector for the custom metric.
 * 16. Describe current anomaly detectors.
 * 17. Get a metric image for the custom metric.
 * 18. Clean up the Amazon CloudWatch resources.
 */
public class CloudWatchScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        final String usage = """

                Usage:
                  <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage> \s

                Where:
                  myDate - The start date to use to get metric statistics. (For
 example, 2023-01-11T18:35:24.00Z.)\s
```

```
                costDateWeek - The start date to use to get AWS/Billinget
statistics. (For example, 2023-01-11T18:35:24.00Z.)\s
                dashboardName - The name of the dashboard to create.\s
                dashboardJson - The location of a JSON file to use to create a
dashboard. (See Readme file.)\s
                dashboardAdd - The location of a JSON file to use to update a
dashboard. (See Readme file.)\s
                settings - The location of a JSON file from which various
values are read. (See Readme file.)\s
                metricImage - The location of a BMP file that is used to create
a graph.\s
               """;

       if (args.length != 7) {
           System.out.println(usage);
           System.exit(1);
       }

       Region region = Region.US_EAST_1;
       String myDate = args[0];
       String costDateWeek = args[1];
       String dashboardName = args[2];
       String dashboardJson = args[3];
       String dashboardAdd = args[4];
       String settings = args[5];
       String metricImage = args[6];

       Double dataPoint = Double.parseDouble("10.0");
       Scanner sc = new Scanner(System.in);
       CloudWatchClient cw = CloudWatchClient.builder()
               .region(region)
               .credentialsProvider(ProfileCredentialsProvider.create())
               .build();

       System.out.println(DASHES);
       System.out.println("Welcome to the Amazon CloudWatch example scenario.");
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println(
               "1. List at least five available unique namespaces from Amazon
CloudWatch. Select one from the list.");
       ArrayList<String> list = listNameSpaces(cw);
       for (int z = 0; z < 5; z++) {
```

```
            int index = z + 1;
            System.out.println("    " + index + ". " + list.get(z));
        }

        String selectedNamespace = "";
        String selectedMetrics = "";
        int num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            selectedNamespace = list.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + selectedNamespace);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. List available metrics within the selected
namespace and select one from the list.");
        ArrayList<String> metList = listMets(cw, selectedNamespace);
        for (int z = 0; z < 5; z++) {
            int index = z + 1;
            System.out.println("    " + index + ". " + metList.get(z));
        }
        num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            selectedMetrics = metList.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + selectedMetrics);
        Dimension myDimension = getSpecificMet(cw, selectedNamespace);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get statistics for the selected metric over the
last day.");
        String metricOption = "";
        ArrayList<String> statTypes = new ArrayList<>();
        statTypes.add("SampleCount");
        statTypes.add("Average");
        statTypes.add("Sum");
        statTypes.add("Minimum");
```

```
        statTypes.add("Maximum");

        for (int t = 0; t < 5; t++) {
            System.out.println("    " + (t + 1) + ". " + statTypes.get(t));
        }
        System.out.println("Select a metric statistic by entering a number from
the preceding list:");
        num = Integer.parseInt(sc.nextLine());
        if (1 <= num && num <= 5) {
            metricOption = statTypes.get(num - 1);
        } else {
            System.out.println("You did not select a valid option.");
            System.exit(1);
        }
        System.out.println("You selected " + metricOption);
        getAndDisplayMetricStatistics(cw, selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get CloudWatch estimated billing for the last
week.");
        getMetricStatistics(cw, costDateWeek);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Create a new CloudWatch dashboard with metrics.");
        createDashboardWithMetrics(cw, dashboardName, dashboardJson);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. List dashboards using a paginator.");
        listDashboards(cw);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Create a new custom metric by adding data to
it.");
        createNewCustomMetric(cw, dataPoint);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Add an additional metric to the dashboard.");
        addMetricToDashboard(cw, dashboardAdd, dashboardName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Create an alarm for the custom metric.");
        String alarmName = createAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Describe ten current alarms.");
        describeAlarms(cw);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Get current data for new custom metric.");
        getCustomMetricData(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Push data into the custom metric to trigger the
alarm.");
        addMetricDataForAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Check the alarm state using the action
DescribeAlarmsForMetric.");
        checkForMetricAlarm(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Get alarm history for the new alarm.");
        getAlarmHistory(cw, settings, myDate);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Add an anomaly detector for the custom metric.");
        addAnomalyDetector(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Describe current anomaly detectors.");
        describeAnomalyDetectors(cw, settings);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("17. Get a metric image for the custom metric.");
        getAndOpenMetricImage(cw, metricImage);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("18. Clean up the Amazon CloudWatch resources.");
        deleteDashboard(cw, dashboardName);
        deleteCWAlarm(cw, alarmName);
        deleteAnomalyDetector(cw, settings);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Amazon CloudWatch example scenario is
complete.");
        System.out.println(DASHES);
        cw.close();
    }

    public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

            DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();
```

```
            cw.deleteAnomalyDetector(request);
            System.out.println("Successfully deleted the Anomaly Detector.");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
        try {
            DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.deleteAlarms(request);
            System.out.println("Successfully deleted alarm " + alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
        try {
            DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
                    .dashboardNames(dashboardName)
                    .build();
            cw.deleteDashboards(dashboardsRequest);
            System.out.println(dashboardName + " was successfully deleted.");

        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
```

```
        System.out.println("Getting Image data for custom metric.");
        try {
            String myJSON = "{\n" +
                    "  \"title\": \"Example Metric Graph\",\n" +
                    "  \"view\": \"timeSeries\",\n" +
                    "  \"stacked \": false,\n" +
                    "  \"period\": 10,\n" +
                    "  \"width\": 1400,\n" +
                    "  \"height\": 600,\n" +
                    "  \"metrics\": [\n" +
                    "    [\n" +
                    "      \"AWS/Billing\",\n" +
                    "      \"EstimatedCharges\",\n" +
                    "      \"Currency\",\n" +
                    "      \"USD\"\n" +
                    "    ]\n" +
                    "  ]\n" +
                    "}";

            GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
                    .metricWidget(myJSON)
                    .build();

            GetMetricWidgetImageResponse response =
cw.getMetricWidgetImage(imageRequest);
            SdkBytes sdkBytes = response.metricWidgetImage();
            byte[] bytes = sdkBytes.asByteArray();
            File outputFile = new File(fileName);
            try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                outputStream.write(bytes);
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
        try {
            // Read values from the JSON file.
```

```
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                    .maxResults(10)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
            List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
            for (AnomalyDetector detector : anomalyDetectorList) {
                System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
                System.out.println("State: " + detector.stateValue());
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
```

```
            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

            PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

            cw.putAnomalyDetector(anomalyDetectorRequest);
            System.out.println("Added anomaly detector for metric " +
customMetricName + ".");

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String alarmName = rootNode.findValue("exampleAlarmName").asText();

            Instant start = Instant.parse(date);
            Instant endDate = Instant.now();
            DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                    .startDate(start)
                    .endDate(endDate)
                    .alarmName(alarmName)
                    .historyItemType(HistoryItemType.ACTION)
                    .build();

            DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
```

```
            List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
            if (historyItems.isEmpty()) {
                System.out.println("No alarm history data found for " + alarmName
+ ".");
            } else {
                for (AlarmHistoryItem item : historyItems) {
                    System.out.println("History summary: " +
item.historySummary());
                    System.out.println("Time stamp: " + item.timestamp());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            boolean hasAlarm = false;
            int retries = 10;

            DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            while (!hasAlarm && retries > 0) {
                DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
                hasAlarm = response.hasMetricAlarms();
                retries--;
```

```
                Thread.sleep(20000);
                System.out.println(".");
            }
            if (!hasAlarm)
                System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
            else
                System.out.println("Alarm state found for " + customMetricName +
".");

        } catch (CloudWatchException | IOException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            // Set an Instant object.
            String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
            Instant instant = Instant.parse(time);

            MetricDatum datum = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

            MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
```

```
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

            List<MetricDatum> metricDataList = new ArrayList<>();
            metricDataList.add(datum);
            metricDataList.add(datum2);

            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric " +
customMetricName);

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            // Set the date.
            Instant nowDate = Instant.now();

            long hours = 1;
            long minutes = 30;
            Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
                    ChronoUnit.MINUTES);
```

```
            Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

            MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(1)
                    .metric(met)
                    .build();

            MetricDataQuery dataQUery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

            List<MetricDataQuery> dq = new ArrayList<>();
            dq.add(dataQUery);

            GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(date2)
                    .metricDataQueries(dq)
                    .build();

            GetMetricDataResponse response = cw.getMetricData(getMetReq);
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                System.out.println("The label is " + item.label());
                System.out.println("The status code is " +
item.statusCode().toString());
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void describeAlarms(CloudWatchClient cw) {
        try {
```

```
            List<AlarmType> typeList = new ArrayList<>();
            typeList.add(AlarmType.METRIC_ALARM);

            DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
                    .alarmTypes(typeList)
                    .maxRecords(10)
                    .build();

            DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
            List<MetricAlarm> alarmList = response.metricAlarms();
            for (MetricAlarm alarm : alarmList) {
                System.out.println("Alarm name: " + alarm.alarmName());
                System.out.println("Alarm description: " +
alarm.alarmDescription());
            }
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static String createAlarm(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            String alarmName = rootNode.findValue("exampleAlarmName").asText();
            String emailTopic = rootNode.findValue("emailTopic").asText();
            String accountId = rootNode.findValue("accountId").asText();
            String region = rootNode.findValue("region").asText();

            // Create a List for alarm actions.
            List<String> alarmActions = new ArrayList<>();
            alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
            PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
                    .alarmActions(alarmActions)
                    .alarmDescription("Example metric alarm")
```

```
                    .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
                    .threshold(100.00)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .evaluationPeriods(1)
                    .period(10)
                    .statistic("Maximum")
                    .datapointsToAlarm(1)
                    .treatMissingData("ignore")
                    .build();

            cw.putMetricAlarm(alarmRequest);
            System.out.println(alarmName + " was successfully created!");
            return alarmName;

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static void addMetricToDashboard(CloudWatchClient cw, String fileName,
String dashboardName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully updated.");

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void createNewCustomMetric(CloudWatchClient cw, Double
dataPoint) {
        try {
```

```
            Dimension dimension = Dimension.builder()
                    .name("UNIQUE_PAGES")
                    .value("URLS")
                    .build();

            // Set an Instant object.
            String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
            Instant instant = Instant.parse(time);

            MetricDatum datum = MetricDatum.builder()
                    .metricName("PAGES_VISITED")
                    .unit(StandardUnit.NONE)
                    .value(dataPoint)
                    .timestamp(instant)
                    .dimensions(dimension)
                    .build();

            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace("SITE/TRAFFIC")
                    .metricData(datum)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric
PAGES_VISITED");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listDashboards(CloudWatchClient cw) {
        try {
            ListDashboardsIterable listRes = cw.listDashboardsPaginator();
            listRes.stream()
                    .flatMap(r -> r.dashboardEntries().stream())
                    .forEach(entry -> {
                        System.out.println("Dashboard name is: " +
entry.dashboardName());
                        System.out.println("Dashboard ARN is: " +
entry.dashboardArn());
                    });
```

```
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createDashboardWithMetrics(CloudWatchClient cw, String
dashboardName, String fileName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            PutDashboardResponse response = cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully created.");
            List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
            if (messages.isEmpty()) {
                System.out.println("There are no messages in the new Dashboard");
            } else {
                for (DashboardValidationMessage message : messages) {
                    System.out.println("Message is: " + message.message());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String readFileAsString(String file) throws IOException {
        return new String(Files.readAllBytes(Paths.get(file)));
    }

    public static void getMetricStatistics(CloudWatchClient cw, String
costDateWeek) {
        try {
            Instant start = Instant.parse(costDateWeek);
            Instant endDate = Instant.now();
            Dimension dimension = Dimension.builder()
                    .name("Currency")
```

```
                    .value("USD")
                    .build();

            List<Dimension> dimensionList = new ArrayList<>();
            dimensionList.add(dimension);
            GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
                    .metricName("EstimatedCharges")
                    .namespace("AWS/Billing")
                    .dimensions(dimensionList)
                    .statistics(Statistic.MAXIMUM)
                    .startTime(start)
                    .endTime(endDate)
                    .period(86400)
                    .build();

            GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
            List<Datapoint> data = response.datapoints();
            if (!data.isEmpty()) {
                for (Datapoint datapoint : data) {
                    System.out
                            .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
                }
            } else {
                System.out.println("The returned data list is empty");
            }

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
nameSpace, String metVal,
            String metricOption, String date, Dimension myDimension) {
        try {
            Instant start = Instant.parse(date);
            Instant endDate = Instant.now();

            GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
```

```
                    .endTime(endDate)
                    .startTime(start)
                    .dimensions(myDimension)
                    .metricName(metVal)
                    .namespace(nameSpace)
                    .period(86400)
                    .statistics(Statistic.fromValue(metricOption))
                    .build();

            GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
            List<Datapoint> data = response.datapoints();
            if (!data.isEmpty()) {
                for (Datapoint datapoint : data) {
                    System.out
                            .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
                }
            } else {
                System.out.println("The returned data list is empty");
            }

        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static Dimension getSpecificMet(CloudWatchClient cw, String namespace)
{
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsResponse response = cw.listMetrics(request);
            List<Metric> myList = response.metrics();
            Metric metric = myList.get(0);
            return metric.dimensions().get(0);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
        return null;
    }

    public static ArrayList<String> listMets(CloudWatchClient cw, String
namespace) {
        try {
            ArrayList<String> metList = new ArrayList<>();
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> metList.add(metrics.metricName()));

            return metList;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public static ArrayList<String> listNameSpaces(CloudWatchClient cw) {
        try {
            ArrayList<String> nameSpaceList = new ArrayList<>();
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
                    .forEach(metrics -> {
                        String data = metrics.namespace();
                        if (!nameSpaceList.contains(data)) {
                            nameSpaceList.add(data);
                        }
                    });

            return nameSpaceList;
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
                System.exit(1);
            }
            return null;
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的以下主题。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:
 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

 This Kotlin code example performs the following tasks:

 1. List available namespaces from Amazon CloudWatch. Select a namespace from the
 list.
 2. List available metrics within the selected namespace.
 3. Get statistics for the selected metric over the last day.
 4. Get CloudWatch estimated billing for the last week.
 5. Create a new CloudWatch dashboard with metrics.
 6. List dashboards using a paginator.
 7. Create a new custom metric by adding data for it.
 8. Add the custom metric to the dashboard.
 9. Create an alarm for the custom metric.
 10. Describe current alarms.
 11. Get current data for the new custom metric.
 12. Push data into the custom metric to trigger the alarm.
 13. Check the alarm state using the action DescribeAlarmsForMetric.
 14. Get alarm history for the new alarm.
 15. Add an anomaly detector for the custom metric.
 16. Describe current anomaly detectors.
 17. Get a metric image for the custom metric.
 18. Clean up the Amazon CloudWatch resources.
 */

val DASHES: String? = String(CharArray(80)).replace("\u0000", "-")

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <myDate> <costDateWeek> <dashboardName> <dashboardJson>
 <dashboardAdd> <settings> <metricImage>

        Where:
```

```
            myDate - The start date to use to get metric statistics. (For
example, 2023-01-11T18:35:24.00Z.)
            costDateWeek - The start date to use to get AWS Billing and Cost
Management statistics. (For example, 2023-01-11T18:35:24.00Z.)
            dashboardName - The name of the dashboard to create.
            dashboardJson - The location of a JSON file to use to create a
dashboard. (See Readme file.)
            dashboardAdd - The location of a JSON file to use to update a
dashboard. (See Readme file.)
            settings - The location of a JSON file from which various values are
read. (See Readme file.)
            metricImage - The location of a BMP file that is used to create a
graph.
    """

    if (args.size != 7) {
        println(usage)
        System.exit(1)
    }

    val myDate = args[0]
    val costDateWeek = args[1]
    val dashboardName = args[2]
    val dashboardJson = args[3]
    val dashboardAdd = args[4]
    val settings = args[5]
    var metricImage = args[6]
    val dataPoint = "10.0".toDouble()
    val inOb = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the Amazon CloudWatch example scenario.")
    println(DASHES)

    println(DASHES)
    println("1. List at least five available unique namespaces from Amazon
CloudWatch. Select a CloudWatch namespace from the list.")
    val list: ArrayList<String> = listNameSpaces()
    for (z in 0..4) {
        println("    ${z + 1}. ${list[z]}")
    }

    var selectedNamespace: String
    var selectedMetrics = ""
```

```
    var num = inOb.nextLine().toInt()
    println("You selected $num")

    if (1 <= num && num <= 5) {
        selectedNamespace = list[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $selectedNamespace")
    println(DASHES)

    println(DASHES)
    println("2. List available metrics within the selected namespace and select
one from the list.")
    val metList = listMets(selectedNamespace)
    for (z in 0..4) {
        println("    ${ z + 1}. ${metList?.get(z)}")
    }
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        selectedMetrics = metList!![num - 1]
    } else {
        println("You did not select a valid option.")
        System.exit(1)
    }
    println("You selected $selectedMetrics")
    val myDimension = getSpecificMet(selectedNamespace)
    if (myDimension == null) {
        println("Error - Dimension is null")
        exitProcess(1)
    }
    println(DASHES)

    println(DASHES)
    println("3. Get statistics for the selected metric over the last day.")
    val metricOption: String
    val statTypes = ArrayList<String>()
    statTypes.add("SampleCount")
    statTypes.add("Average")
    statTypes.add("Sum")
    statTypes.add("Minimum")
    statTypes.add("Maximum")
```

```
    for (t in 0..4) {
        println("    ${t + 1}. ${statTypes[t]}")
    }
    println("Select a metric statistic by entering a number from the preceding
list:")
    num = inOb.nextLine().toInt()
    if (1 <= num && num <= 5) {
        metricOption = statTypes[num - 1]
    } else {
        println("You did not select a valid option.")
        exitProcess(1)
    }
    println("You selected $metricOption")
    getAndDisplayMetricStatistics(selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension)
    println(DASHES)

    println(DASHES)
    println("4. Get CloudWatch estimated billing for the last week.")
    getMetricStatistics(costDateWeek)
    println(DASHES)

    println(DASHES)
    println("5. Create a new CloudWatch dashboard with metrics.")
    createDashboardWithMetrics(dashboardName, dashboardJson)
    println(DASHES)

    println(DASHES)
    println("6. List dashboards using a paginator.")
    listDashboards()
    println(DASHES)

    println(DASHES)
    println("7. Create a new custom metric by adding data to it.")
    createNewCustomMetric(dataPoint)
    println(DASHES)

    println(DASHES)
    println("8. Add an additional metric to the dashboard.")
    addMetricToDashboard(dashboardAdd, dashboardName)
    println(DASHES)

    println(DASHES)
    println("9. Create an alarm for the custom metric.")
```

```
    val alarmName: String = createAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("10. Describe 10 current alarms.")
    describeAlarms()
    println(DASHES)

    println(DASHES)
    println("11. Get current data for the new custom metric.")
    getCustomMetricData(settings)
    println(DASHES)

    println(DASHES)
    println("12. Push data into the custom metric to trigger the alarm.")
    addMetricDataForAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("13. Check the alarm state using the action
 DescribeAlarmsForMetric.")
    checkForMetricAlarm(settings)
    println(DASHES)

    println(DASHES)
    println("14. Get alarm history for the new alarm.")
    getAlarmHistory(settings, myDate)
    println(DASHES)

    println(DASHES)
    println("15. Add an anomaly detector for the custom metric.")
    addAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("16. Describe current anomaly detectors.")
    describeAnomalyDetectors(settings)
    println(DASHES)

    println(DASHES)
    println("17. Get a metric image for the custom metric.")
    getAndOpenMetricImage(metricImage)
    println(DASHES)
```

```
    println(DASHES)
    println("18. Clean up the Amazon CloudWatch resources.")
    deleteDashboard(dashboardName)
    deleteAlarm(alarmName)
    deleteAnomalyDetector(settings)
    println(DASHES)

    println(DASHES)
    println("The Amazon CloudWatch example scenario is complete.")
    println(DASHES)
}

suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}

suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }
```

```
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}

suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}

suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
        ]
        }"""

    val imageRequest =
        GetMetricWidgetImageRequest {
            metricWidget = myJSON
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
```

```
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}

suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}

suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
```

```
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}

suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest =
        DescribeAlarmHistoryRequest {
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDateVal)
            alarmName = alarmNameVal
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
```

```
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}

suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
        if (!hasAlarm) {
            println("No Alarm state found for $customMetricName after 10
 retries.")
        } else {
            println("Alarm state found for $customMetricName.")
        }
    }
}
```

```
suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
```

```
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}

suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
        MetricStat {
            stat = "Maximum"
            period = 1
            metric = met
        }

    val dataQUery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
```

```
    dq.add(dataQUery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
            scanBy = ScanBy.TimestampDescending
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(nowDate)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(date2)
            metricDataQueries = dq
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}

suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}

suspend fun createAlarm(fileName: String): String {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
```

```
    val rootNode: JsonNode = ObjectMapper().readTree(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val emailTopic = rootNode.findValue("emailTopic").asText()
    val accountId = rootNode.findValue("accountId").asText()
    val region2 = rootNode.findValue("region").asText()

    // Create a List for alarm actions.
    val alarmActionObs: MutableList<String> = ArrayList()
    alarmActionObs.add("arn:aws:sns:$region2:$accountId:$emailTopic")
    val alarmRequest =
        PutMetricAlarmRequest {
            alarmActions = alarmActionObs
            alarmDescription = "Example metric alarm"
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanOrEqualToThreshold
            threshold = 100.00
            metricName = customMetricName
            namespace = customMetricNamespace
            evaluationPeriods = 1
            period = 10
            statistic = Statistic.Maximum
            datapointsToAlarm = 1
            treatMissingData = "ignore"
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(alarmRequest)
        println("$alarmNameVal was successfully created!")
        return alarmNameVal
    }
}

suspend fun addMetricToDashboard(
    fileNameVal: String,
    dashboardNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }
```

```
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully updated.")
    }
}

suspend fun createNewCustomMetric(dataPoint: Double) {
    val dimension =
        Dimension {
            name = "UNIQUE_PAGES"
            value = "URLS"
        }

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = "PAGES_VISITED"
            unit = StandardUnit.None
            value = dataPoint
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
            dimensions = listOf(dimension)
        }

    val request =
        PutMetricDataRequest {
            namespace = "SITE/TRAFFIC"
            metricData = listOf(datum)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric PAGES_VISITED")
    }
}

suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
```

```
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}

suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}

fun readFileAsString(file: String): String =
 String(Files.readAllBytes(Paths.get(file)))

suspend fun getMetricStatistics(costDateWeek: String?) {
    val start = Instant.parse(costDateWeek)
    val endDate = Instant.now()
    val dimension =
        Dimension {
            name = "Currency"
```

```
            value = "USD"
        }

    val dimensionList: MutableList<Dimension> = ArrayList()
    dimensionList.add(dimension)

    val statisticsRequest =
        GetMetricStatisticsRequest {
            metricName = "EstimatedCharges"
            namespace = "AWS/Billing"
            dimensions = dimensionList
            statistics = listOf(Statistic.Maximum)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            period = 86400
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data: List<Datapoint>? = response.datapoints
        if (data != null) {
            if (!data.isEmpty()) {
                for (datapoint in data) {
                    println("Timestamp:  ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
```

```
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
```

```
            }
        }
    }
    return metList
}

suspend fun getSpecificMet(namespaceVal: String?): Dimension? {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(request)
        val myList = response.metrics
        if (myList != null) {
            return myList[0].dimensions?.get(0)
        }
    }
    return null
}

suspend fun listNameSpaces(): ArrayList<String> {
    val nameSpaceList = ArrayList<String>()
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(ListMetricsRequest {})
        response.metrics?.forEach { metrics ->
            val data = metrics.namespace
            if (!nameSpaceList.contains(data)) {
                nameSpaceList.add(data!!)
            }
        }
    }
    return nameSpaceList
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的以下主题。
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory

- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 的 CloudWatch 操作

以下代码示例演示了如何使用 AWS SDK 来执行单个 CloudWatch 操作。每个示例都包含一个指向 GitHub 的链接，您可以在其中找到有关设置和运行代码的说明。

这些代码节选将调用 CloudWatch API，是必须在上下文中运行的大型程序的代码节选。您可以在使用 AWS SDK 的 CloudWatch 场景 中结合上下文查看操作。

以下示例仅包括最常用的操作。有关完整列表，请参阅 Amazon CloudWatch API 参考。

示例

- 将 DeleteAlarms 与 AWS SDK 或 CLI 配合使用
- 将 DeleteAnomalyDetector 与 AWS SDK 或 CLI 配合使用
- 将 DeleteDashboards 与 AWS SDK 或 CLI 配合使用
- 将 DescribeAlarmHistory 与 AWS SDK 或 CLI 配合使用
- 将 DescribeAlarms 与 AWS SDK 或 CLI 配合使用
- 将 DescribeAlarmsForMetric 与 AWS SDK 或 CLI 配合使用
- 将 DescribeAnomalyDetectors 与 AWS SDK 或 CLI 配合使用
- 将 DisableAlarmActions 与 AWS SDK 或 CLI 配合使用

- 将 EnableAlarmActions 与 AWS SDK 或 CLI 配合使用
- 将 GetDashboard 与 AWS SDK 或 CLI 配合使用
- 将 GetMetricData 与 AWS SDK 或 CLI 配合使用
- 将 GetMetricStatistics 与 AWS SDK 或 CLI 配合使用
- 将 GetMetricWidgetImage 与 AWS SDK 或 CLI 配合使用
- 将 ListDashboards 与 AWS SDK 或 CLI 配合使用
- 将 ListMetrics 与 AWS SDK 或 CLI 配合使用
- 将 PutAnomalyDetector 与 AWS SDK 或 CLI 配合使用
- 将 PutDashboard 与 AWS SDK 或 CLI 配合使用
- 将 PutMetricAlarm 与 AWS SDK 或 CLI 配合使用
- 将 PutMetricData 与 AWS SDK 或 CLI 配合使用

将 **DeleteAlarms** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteAlarms。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 开始使用告警
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
```

```
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DeleteAlarms。

C++

适用于 C++ 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <iostream>
```

删除告警。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DeleteAlarmsRequest request;
        request.AddAlarmNames(alarm_name);
```

```
        auto outcome = cw.DeleteAlarms(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to delete CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 DeleteAlarms。

CLI

AWS CLI

删除警报

以下示例使用 `delete-alarms` 命令删除名为“myalarm”的 Amazon CloudWatch 警报：

```
aws cloudwatch delete-alarms --alarm-names myalarm
```

输出：

```
This command returns to the prompt if successful.
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 DeleteAlarms。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteAlarm {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to delete (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_2;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        deleteCWAlarm(cw, alarmName);
        cw.close();
    }

    public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
        try {
```

```
            DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.deleteAlarms(request);
            System.out.printf("Successfully deleted alarm %s", alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteAlarms。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 DeleteAlarms。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
// 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);
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的 DeleteAlarms。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteAlarms。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
 metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DeleteAlarms。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
TRY.
    lo_cwt->deletealarms(
      it_alarmnames = it_alarm_names
    ).
    MESSAGE 'Alarms deleted.' TYPE 'I'.
  CATCH /aws1/cx_cwtresourcenotfound .
    MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 DeleteAlarms。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DeleteAnomalyDetector** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteAnomalyDetector。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DeleteAnomalyDetector。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .stat("Maximum")
                   .build();

           DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                   .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                   .build();

           cw.deleteAnomalyDetector(request);
           System.out.println("Successfully deleted the Anomaly Detector.");

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteAnomalyDetector。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the Anomaly Detector.")
    }
}
```

- 有关 API 的详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteAnomalyDetector。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DeleteDashboards** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteDashboards。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DeleteDashboards。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
       try {
           DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
                   .dashboardNames(dashboardName)
                   .build();
           cw.deleteDashboards(dashboardsRequest);
           System.out.println(dashboardName + " was successfully deleted.");

       } catch (CloudWatchException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DeleteDashboards。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun deleteDashboard(dashboardName: String) {
```

```
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DeleteDashboards。

PowerShell

适用于 PowerShell 的工具

示例 1：删除指定的控制面板，继续操作前提示确认。要绕过确认，请在命令中添加 -Force 开关。

```
Remove-CWDashboard -DashboardName Dashboard1
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 DeleteDashboards。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeAlarmHistory** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeAlarmHistory。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DescribeAlarmHistory。

CLI

AWS CLI

检索警报的历史记录

以下示例使用 `describe-alarm-history` 命令检索名为“myalarm”的 Amazon CloudWatch 警报的历史记录：

```
aws cloudwatch describe-alarm-history --alarm-name "myalarm" --history-item-
type StateUpdate
```

输出：

```
{
    "AlarmHistoryItems": [
        {
            "Timestamp": "2014-04-09T18:59:06.442Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"ALARM\",\"stateReason\":\"testing purposes\"},\"newState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were not
 greater than the threshold (70.0). The most recent datapoints: [38.958,
 40.292].\",\"stateReasonData\":{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}}}",
            "HistorySummary": "Alarm updated from ALARM to OK"
        },
        {
            "Timestamp": "2014-04-09T18:59:05.805Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were
 not greater than the threshold (70.0). The most recent datapoints:
 [38.839999999999996, 39.714].\",\"stateReasonData\":{\"version\":
\"1.0\",\"queryDate\":\"2014-03-11T22:45:41.569+0000\",\"startDate\":
\"2014-03-11T22:30:00.000+0000\",\"statistic\":\"Average\",\"period\":300,
\"recentDatapoints\":[38.839999999999996,39.714],\"threshold\":70.0}},\"newState
\":{\"stateValue\":\"ALARM\",\"stateReason\":\"testing purposes\"}}",
            "HistorySummary": "Alarm updated from OK to ALARM"
```

```
        }
    ]
}
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 DescribeAlarmHistory。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String alarmName = rootNode.findValue("exampleAlarmName").asText();

           Instant start = Instant.parse(date);
           Instant endDate = Instant.now();
           DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                   .startDate(start)
                   .endDate(endDate)
                   .alarmName(alarmName)
                   .historyItemType(HistoryItemType.ACTION)
                   .build();

           DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
           List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
           if (historyItems.isEmpty()) {
               System.out.println("No alarm history data found for " + alarmName
+ ".");
```

```
            } else {
                for (AlarmHistoryItem item : historyItems) {
                    System.out.println("History summary: " +
 item.historySummary());
                    System.out.println("Time stamp: " + item.timestamp());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeAlarmHistory。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest =
        DescribeAlarmHistoryRequest {
```

```
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDateVal)
            alarmName = alarmNameVal
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》https://sdk.amazonaws.com/kotlin/api/latest/index.html中的 DescribeAlarmHistory。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeAlarms** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeAlarms。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 开始使用告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   /// <summary>
   /// Describe the current alarms, optionally filtered by state.
   /// </summary>
   /// <param name="stateValue">Optional filter for alarm state.</param>
   /// <returns>The list of alarm data.</returns>
   public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
   {
       List<MetricAlarm> alarms = new List<MetricAlarm>();
       var paginatedDescribeAlarms =
_amazonCloudWatch.Paginators.DescribeAlarms(
           new DescribeAlarmsRequest()
           {
               StateValue = stateValue
           });

       await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
       {
           alarms.Add(data);
       }
       return alarms;
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DescribeAlarms。

CLI

AWS CLI

列出有关警报的信息

以下示例使用 describe-alarms 命令提供名为“myalarm”的警报的相关信息：

```
aws cloudwatch describe-alarms --alarm-names "myalarm"
```

输出：

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:myalarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2012-12-27T00:49:54.032Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:123456789012:myHighCpuAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myalarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
```

```
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        }
    ]
}
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 DescribeAlarms。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void describeAlarms(CloudWatchClient cw) {
        try {
            List<AlarmType> typeList = new ArrayList<>();
            typeList.add(AlarmType.METRIC_ALARM);

            DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
                    .alarmTypes(typeList)
                    .maxRecords(10)
                    .build();

            DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
            List<MetricAlarm> alarmList = response.metricAlarms();
            for (MetricAlarm alarm : alarmList) {
                System.out.println("Alarm name: " + alarm.alarmName());
                System.out.println("Alarm description: " +
 alarm.alarmDescription());
            }
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeAlarms。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeAlarms。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
require "aws-sdk-cloudwatch"

# Lists the names of available Amazon CloudWatch alarms.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   list_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def list_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts alarm.alarm_name
    end
  else
    puts "No alarms found."
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 DescribeAlarms。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    TRY.
        oo_result = lo_cwt->describealarms(                " oo_result is
 returned for testing purposes. "
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarms retrieved.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 DescribeAlarms。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeAlarmsForMetric** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeAlarmsForMetric。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DescribeAlarmsForMetric。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

描述告警。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DescribeAlarmsRequest request;
        request.SetMaxRecords(1);

        bool done = false;
        bool header = false;
        while (!done)
        {
            auto outcome = cw.DescribeAlarms(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to describe CloudWatch alarms:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left <<
                    std::setw(32) << "Name" <<
```

```
                std::setw(64) << "Arn" <<
                std::setw(64) << "Description" <<
                std::setw(20) << "LastUpdated" <<
                std::endl;
            header = true;
        }

        const auto &alarms = outcome.GetResult().GetMetricAlarms();
        for (const auto &alarm : alarms)
        {
            std::cout << std::left <<
                std::setw(32) << alarm.GetAlarmName() <<
                std::setw(64) << alarm.GetAlarmArn() <<
                std::setw(64) << alarm.GetAlarmDescription() <<
                std::setw(20) <<
                alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
                    SIMPLE_DATE_FORMAT_STR) <<
                std::endl;
        }

        const auto &next_token = outcome.GetResult().GetNextToken();
        request.SetNextToken(next_token);
        done = next_token.empty();
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 DescribeAlarmsForMetric。

CLI

AWS CLI

显示与指标关联的警报的相关信息

以下示例使用 `describe-alarms-for-metric` 命令显示与 Amazon EC2 CPUUtilization 指标和 ID 为 i-0c986c72 的实例关联的任何警报的相关信息：

```
aws cloudwatch describe-alarms-for-metric --metric-name CPUUtilization --
namespace AWS/EC2 --dimensions Name=InstanceId,Value=i-0c986c72
```

输出：

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 10,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm2",
            "StateUpdatedTimestamp": "2013-10-30T03:03:51.479Z",
            "AlarmConfigurationUpdatedTimestamp": "2013-10-30T03:03:50.865Z",
            "ComparisonOperator": "GreaterThanOrEqualToThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:111122223333:NotifyMe"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2013-10-30T03:03:51.479+0000\",\"startDate\":\"2013-10-30T02:08:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":
[40.698,39.612,42.432,39.796,38.816,42.28,42.854,40.088,40.760000000000005,41.316],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm2",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 10 datapoints were not
 greater than or equal to the threshold (70.0). The most recent datapoints:
 [40.760000000000005, 41.316].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        },
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
```

```
            "AlarmConfigurationUpdatedTimestamp": "2014-04-09T22:26:05.958Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-east-1:111122223333:HighCPUAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": false,
            "MetricName": "CPUUtilization"
        }
    ]
}
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 DescribeAlarmsForMetric。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
   public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
           boolean hasAlarm = false;
           int retries = 10;

           DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           while (!hasAlarm && retries > 0) {
               DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
               hasAlarm = response.hasMetricAlarms();
               retries--;
               Thread.sleep(20000);
               System.out.println(".");
           }
           if (!hasAlarm)
               System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
```

```
            else
                System.out.println("Alarm state found for " + customMetricName +
 ".");

        } catch (CloudWatchException | IOException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeAlarmsForMetric。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 DescribeAlarmsForMetric。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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);
    });
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的 DescribeAlarmsForMetric。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
```

```
        if (!hasAlarm) {
            println("No Alarm state found for $customMetricName after 10
 retries.")
        } else {
            println("Alarm state found for $customMetricName.")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DescribeAlarmsForMetric。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
```

```
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
 metric_name)
        return alarm_iter
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DescribeAlarmsForMetric。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   describe_metric_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def describe_metric_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms

  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts "-" * 16
      puts "Name:           " + alarm.alarm_name
      puts "State value:    " + alarm.state_value
      puts "State reason:   " + alarm.state_reason
      puts "Metric:         " + alarm.metric_name
      puts "Namespace:      " + alarm.namespace
      puts "Statistic:      " + alarm.statistic
      puts "Period:         " + alarm.period.to_s
      puts "Unit:           " + alarm.unit.to_s
```

```
      puts "Eval. periods:  " + alarm.evaluation_periods.to_s
      puts "Threshold:      " + alarm.threshold.to_s
      puts "Comp. operator: " + alarm.comparison_operator

      if alarm.key?(:ok_actions) && alarm.ok_actions.count.positive?
        puts "OK actions:"
        alarm.ok_actions.each do |a|
          puts "  " + a
        end
      end

      if alarm.key?(:alarm_actions) && alarm.alarm_actions.count.positive?
        puts "Alarm actions:"
        alarm.alarm_actions.each do |a|
          puts "  " + a
        end
      end

      if alarm.key?(:insufficient_data_actions) &&
          alarm.insufficient_data_actions.count.positive?
        puts "Insufficient data actions:"
        alarm.insufficient_data_actions.each do |a|
          puts "  " + a
        end
      end

      puts "Dimensions:"
      if alarm.key?(:dimensions) && alarm.dimensions.count.positive?
        alarm.dimensions.each do |d|
          puts "  Name: " + d.name + ", Value: " + d.value
        end
      else
        puts "  None for this alarm."
      end
    end
  else
    puts "No alarms found."
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end

# Example usage:
def run_me
```

```
  region = ""

  # Print usage information and then stop.
  if ARGV[0] == "--help" || ARGV[0] == "-h"
    puts "Usage:   ruby cw-ruby-example-show-alarms.rb REGION"
    puts "Example: ruby cw-ruby-example-show-alarms.rb us-east-1"
    exit 1
  # If no values are specified at the command prompt, use these default values.
  elsif ARGV.count.zero?
    region = "us-east-1"
  # Otherwise, use the values as specified at the command prompt.
  else
    region = ARGV[0]
  end

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
  puts "Available alarms:"
  describe_metric_alarms(cloudwatch_client)
end

run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 DescribeAlarmsForMetric。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **DescribeAnomalyDetectors** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeAnomalyDetectors。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DescribeAnomalyDetectors。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
           DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                   .maxResults(10)
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
           List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
           for (AnomalyDetector detector : anomalyDetectorList) {
               System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
               System.out.println("State: " + detector.stateValue());
           }

       } catch (CloudWatchException | IOException e) {
           System.err.println(e.getMessage());
           System.exit(1);
```

```
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DescribeAnomalyDetectors。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》https://sdk.amazonaws.com/kotlin/api/latest/index.html中的 DescribeAnomalyDetectors。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 `DisableAlarmActions` 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DisableAlarmActions`。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 开始使用告警
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });
```

```
        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 DisableAlarmActions。

C++

适用于 C++ 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

禁用告警操作。

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::DisableAlarmActionsRequest
 disableAlarmActionsRequest;
        disableAlarmActionsRequest.AddAlarmNames(alarm_name);

        auto disableAlarmActionsOutcome =
 cw.DisableAlarmActions(disableAlarmActionsRequest);
        if (!disableAlarmActionsOutcome.IsSuccess())
        {
            std::cout << "Failed to disable actions for alarm " << alarm_name <<
                ": " << disableAlarmActionsOutcome.GetError().GetMessage() <<
                std::endl;
        }
        else
```

```
        {
            std::cout << "Successfully disabled actions for alarm " <<
                alarm_name << std::endl;
        }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 DisableAlarmActions。

CLI

AWS CLI

禁用警报的操作

以下示例使用 `disable-alarm-actions` 命令禁用名为 myalarm 的警报的所有操作：

```
aws cloudwatch disable-alarm-actions --alarm-names myalarm
```

如果成功，该命令将返回到提示符。

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 DisableAlarmActions。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DisableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to disable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        disableActions(cw, alarmName);
        cw.close();
    }

    public static void disableActions(CloudWatchClient cw, String alarmName) {
        try {
            DisableAlarmActionsRequest request =
 DisableAlarmActionsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.disableAlarmActions(request);
            System.out.printf("Successfully disabled actions on alarm %s",
 alarmName);

        } catch (CloudWatchException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 DisableAlarmActions。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 DisableAlarmActions。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
// 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);
    }
  }
);
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的 DisableAlarmActions。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
suspend fun disableActions(alarmName: String) {
    val request =
        DisableAlarmActionsRequest {
            alarmNames = listOf(alarmName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm $alarmName")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 DisableAlarmActions。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
```

```
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 DisableAlarmActions。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# Disables an alarm in Amazon CloudWatch.
#
# Prerequisites.
#
# - The alarm to disable.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm to disable.
# @return [Boolean] true if the alarm was disabled; otherwise, false.
# @example
#   exit 1 unless alarm_actions_disabled?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket'
#   )
def alarm_actions_disabled?(cloudwatch_client, alarm_name)
  cloudwatch_client.disable_alarm_actions(alarm_names: [alarm_name])
  return true
rescue StandardError => e
  puts "Error disabling alarm actions: #{e.message}"
  return false
end

# Example usage:
def run_me
  alarm_name = "ObjectsInBucket"
  alarm_description = "Objects exist in this bucket for more than 1 day."
  metric_name = "NumberOfObjects"
  # Notify this Amazon Simple Notification Service (Amazon SNS) topic when
  # the alarm transitions to the ALARM state.
  alarm_actions = ["arn:aws:sns:us-
east-1:111111111111:Default_CloudWatch_Alarms_Topic"]
```

```
namespace = "AWS/S3"
statistic = "Average"
dimensions = [
  {
    name: "BucketName",
    value: "doc-example-bucket"
  },
  {
    name: "StorageType",
    value: "AllStorageTypes"
  }
]
period = 86_400 # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
unit = "Count"
evaluation_periods = 1 # More than one day.
threshold = 1 # One object.
comparison_operator = "GreaterThanThreshold" # More than one object.
# Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
region = "us-east-1"

cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

if alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  puts "Alarm '#{alarm_name}' created or updated."
else
  puts "Could not create or update alarm '#{alarm_name}'."
end

if alarm_actions_disabled?(cloudwatch_client, alarm_name)
  puts "Alarm '#{alarm_name}' disabled."
```

```
  else
    puts "Could not disable alarm '#{alarm_name}'."
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 DisableAlarmActions。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    "Disables actions on the specified alarm. "
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarm actions disabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 DisableAlarmActions。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **EnableAlarmActions** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 EnableAlarmActions。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 EnableAlarmActions。

C++

适用于 C++ 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

启用告警操作。

```
    Aws::CloudWatch::CloudWatchClient cw;
    Aws::CloudWatch::Model::PutMetricAlarmRequest request;
    request.SetAlarmName(alarm_name);
    request.SetComparisonOperator(
        Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
    request.SetEvaluationPeriods(1);
    request.SetMetricName("CPUUtilization");
    request.SetNamespace("AWS/EC2");
    request.SetPeriod(60);
    request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
    request.SetThreshold(70.0);
    request.SetActionsEnabled(false);
    request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
    request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
    request.AddAlarmActions(actionArn);

    Aws::CloudWatch::Model::Dimension dimension;
    dimension.SetName("InstanceId");
    dimension.SetValue(instanceId);
    request.AddDimensions(dimension);
```

```
    auto outcome = cw.PutMetricAlarm(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to create CloudWatch alarm:" <<
            outcome.GetError().GetMessage() << std::endl;
        return;
    }

    Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
    enable_request.AddAlarmNames(alarm_name);

    auto enable_outcome = cw.EnableAlarmActions(enable_request);
    if (!enable_outcome.IsSuccess())
    {
        std::cout << "Failed to enable alarm actions:" <<
            enable_outcome.GetError().GetMessage() << std::endl;
        return;
    }

    std::cout << "Successfully created alarm " << alarm_name <<
        " and enabled actions on it." << std::endl;
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 EnableAlarmActions。

CLI

AWS CLI

启用警报的所有操作

以下示例使用 `enable-alarm-actions` 命令启用名为 myalarm 的警报的所有操作：

```
aws cloudwatch enable-alarm-actions --alarm-names myalarm
```

如果成功，该命令将返回到提示符。

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 EnableAlarmActions。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to enable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarm = args[0];
```

```
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        enableActions(cw, alarm);
        cw.close();
    }

    public static void enableActions(CloudWatchClient cw, String alarm) {
        try {
            EnableAlarmActionsRequest request =
 EnableAlarmActionsRequest.builder()
                    .alarmNames(alarm)
                    .build();

            cw.enableAlarmActions(request);
            System.out.printf("Successfully enabled actions on alarm %s", alarm);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 EnableAlarmActions。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 EnableAlarmActions。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
// 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);
      }
    });
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的 EnableAlarmActions。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
suspend fun enableActions(alarm: String) {
    val request =
        EnableAlarmActionsRequest {
            alarmNames = listOf(alarm)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println("Successfully enabled actions on alarm $alarm")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 EnableAlarmActions。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
 they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 EnableAlarmActions。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    "Enable actions on the specified alarm."
    TRY.
        lo_cwt->enablealarmactions(
          it_alarmnames = it_alarm_names
        ).
        MESSAGE 'Alarm actions enabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 EnableAlarmActions。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **GetDashboard** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetDashboard。

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 GetDashboard。

PowerShell

适用于 PowerShell 的工具

示例 1：返回指定控制面板的正文 arn。

```
Get-CWDashboard -DashboardName Dashboard1
```

输出：

```
DashboardArn                                            DashboardBody
```

```
------------                                                  -------------
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 GetDashboard。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **GetMetricData** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetMetricData。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
```

```
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 GetMetricData。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           // Set the date.
           Instant nowDate = Instant.now();

           long hours = 1;
           long minutes = 30;
           Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
                   ChronoUnit.MINUTES);

           Metric met = Metric.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

           MetricStat metStat = MetricStat.builder()
                   .stat("Maximum")
                   .period(1)
                   .metric(met)
                   .build();
```

```
            MetricDataQuery dataQUery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

            List<MetricDataQuery> dq = new ArrayList<>();
            dq.add(dataQUery);

            GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(date2)
                    .metricDataQueries(dq)
                    .build();

            GetMetricDataResponse response = cw.getMetricData(getMetReq);
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                System.out.println("The label is " + item.label());
                System.out.println("The status code is " +
item.statusCode().toString());
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 GetMetricData。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
        MetricStat {
            stat = "Maximum"
            period = 1
            metric = met
        }

    val dataQUery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQUery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
```

```
            scanBy = ScanBy.TimestampDescending
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(nowDate)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(date2)
            metricDataQueries = dq
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 GetMetricData。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **GetMetricStatistics** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetMetricStatistics。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
```

```
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 GetMetricStatistics。

CLI

AWS CLI

获取每个 EC2 实例的 CPU 利用率

以下示例使用 `get-metric-statistics` 命令获取 ID 为 i-abcdef 的 EC2 实例的 CPU 利用率。

```
aws cloudwatch get-metric-statistics --metric-name CPUUtilization --start-
time 2014-04-08T23:18:00Z --end-time 2014-04-09T23:18:00Z --period 3600 --
namespace AWS/EC2 --statistics Maximum --dimensions Name=InstanceId,Value=i-
abcdef
```

输出：

```
{
    "Datapoints": [
        {
```

```
            "Timestamp": "2014-04-09T11:18:00Z",
            "Maximum": 44.79,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T20:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T19:18:00Z",
            "Maximum": 50.85,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T09:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T03:18:00Z",
            "Maximum": 76.84,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T21:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T14:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T08:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T16:18:00Z",
            "Maximum": 45.55,
            "Unit": "Percent"
        },
```

```
        {
            "Timestamp": "2014-04-09T06:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T13:18:00Z",
            "Maximum": 45.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T05:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T18:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T17:18:00Z",
            "Maximum": 52.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T07:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T02:18:00Z",
            "Maximum": 51.23,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T12:18:00Z",
            "Maximum": 47.67,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-08T23:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
```

```
        },
        {
            "Timestamp": "2014-04-09T10:18:00Z",
            "Maximum": 51.91,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T04:18:00Z",
            "Maximum": 47.13,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T15:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T00:18:00Z",
            "Maximum": 48.16,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T01:18:00Z",
            "Maximum": 49.18,
            "Unit": "Percent"
        }
    ],
    "Label": "CPUUtilization"
}
```

指定多个维度

以下示例说明了如何指定多个维度。每个维度都指定为一个“名称/值”对，名称和值之间用逗号隔开。多个维度之间用空格隔开。如果单个指标包含多个维度，则必须为每个已定义的维度指定一个值。

有关使用 `get-metric-statistics` 命令的更多示例，请参阅《Amazon CloudWatch 开发人员指南》中的“获取指标的统计数据”。

```
aws cloudwatch get-metric-statistics --metric-name Buffers --
namespace MyNameSpace --dimensions Name=InstanceID,Value=i-
abcdef Name=InstanceType,Value=m1.small --start-time 2016-10-15T04:00:00Z --end-
time 2016-10-19T07:00:00Z --statistics Average --period 60
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 GetMetricStatistics。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
nameSpace, String metVal,
           String metricOption, String date, Dimension myDimension) {
       try {
           Instant start = Instant.parse(date);
           Instant endDate = Instant.now();

           GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
                   .endTime(endDate)
                   .startTime(start)
                   .dimensions(myDimension)
                   .metricName(metVal)
                   .namespace(nameSpace)
                   .period(86400)
                   .statistics(Statistic.fromValue(metricOption))
                   .build();

           GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
           List<Datapoint> data = response.datapoints();
           if (!data.isEmpty()) {
               for (Datapoint datapoint : data) {
                   System.out
                           .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
               }
           } else {
               System.out.println("The returned data list is empty");
           }
```

```
        } catch (CloudWatchException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 GetMetricStatistics。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
```

```
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 GetMetricStatistics。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
   def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
       """
       Gets statistics for a metric within a specified time span. Metrics are
grouped
       into the specified period.

       :param namespace: The namespace of the metric.
       :param name: The name of the metric.
       :param start: The UTC start time of the time span to retrieve.
       :param end: The UTC end time of the time span to retrieve.
       :param period: The period, in seconds, in which to group metrics. The
period
                      must match the granularity of the metric, which depends on
                      the metric's age. For example, metrics that are older than
                      three hours have a one-minute granularity, so the period
must
                      be at least 60 and must be a multiple of 60.
       :param stat_types: The type of statistics to retrieve, such as average
value
                          or maximum value.
       :return: The retrieved statistics for the metric.
       """
       try:
           metric = self.cloudwatch_resource.Metric(namespace, name)
           stats = metric.get_statistics(
               StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
           )
           logger.info(
               "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
           )
       except ClientError:
           logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
           raise
       else:
           return stats
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 GetMetricStatistics。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **GetMetricWidgetImage** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 GetMetricWidgetImage。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
```

```
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 GetMetricWidgetImage。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
       System.out.println("Getting Image data for custom metric.");
       try {
           String myJSON = "{\n" +
                   "  \"title\": \"Example Metric Graph\",\n" +
                   "  \"view\": \"timeSeries\",\n" +
                   "  \"stacked \": false,\n" +
                   "  \"period\": 10,\n" +
                   "  \"width\": 1400,\n" +
                   "  \"height\": 600,\n" +
                   "  \"metrics\": [\n" +
                   "    [\n" +
                   "      \"AWS/Billing\",\n" +
                   "      \"EstimatedCharges\",\n" +
                   "      \"Currency\",\n" +
                   "      \"USD\"\n" +
                   "    ]\n" +
                   "  ]\n" +
                   "}";

           GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
                   .metricWidget(myJSON)
                   .build();

           GetMetricWidgetImageResponse response =
cw.getMetricWidgetImage(imageRequest);
           SdkBytes sdkBytes = response.metricWidgetImage();
           byte[] bytes = sdkBytes.asByteArray();
           File outputFile = new File(fileName);
```

```
            try (FileOutputStream outputStream = new
 FileOutputStream(outputFile)) {
                outputStream.write(bytes);
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 GetMetricWidgetImage。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
```

```
        ]
        }"""

    val imageRequest =
        GetMetricWidgetImageRequest {
            metricWidget = myJSON
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(imageRequest)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
        }
    }
    println("You have successfully written data to $fileName")
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 GetMetricWidgetImage。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **ListDashboards** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListDashboards`。

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
```

```
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 ListDashboards。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void listDashboards(CloudWatchClient cw) {
        try {
            ListDashboardsIterable listRes = cw.listDashboardsPaginator();
            listRes.stream()
                    .flatMap(r -> r.dashboardEntries().stream())
                    .forEach(entry -> {
                        System.out.println("Dashboard name is: " +
entry.dashboardName());
                        System.out.println("Dashboard ARN is: " +
entry.dashboardArn());
                    });

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 ListDashboards。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 ListDashboards。

PowerShell

适用于 PowerShell 的工具

示例 1：返回您账户的控制面板集合。

```
Get-CWDashboardList
```

输出：

```
DashboardArn DashboardName LastModified        Size
------------ ------------- ------------        ----
arn:...      Dashboard1    7/6/2017 8:14:15 PM 252
```

示例 2：返回名称以前缀“dev”开头的账户的控制面板集合。

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 ListDashboards。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **ListMetrics** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 ListMetrics。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
```

```
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 ListMetrics。

C++

适用于 C++ 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

列出指标。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::ListMetricsRequest request;

        if (argc > 1)
        {
            request.SetMetricName(argv[1]);
        }

        if (argc > 2)
        {
            request.SetNamespace(argv[2]);
        }

        bool done = false;
        bool header = false;
        while (!done)
        {
            auto outcome = cw.ListMetrics(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to list CloudWatch metrics:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left << std::setw(48) << "MetricName" <<
                    std::setw(32) << "Namespace" << "DimensionNameValuePairs" <<
                    std::endl;
                header = true;
            }
```

```
            const auto &metrics = outcome.GetResult().GetMetrics();
            for (const auto &metric : metrics)
            {
                std::cout << std::left << std::setw(48) <<
                    metric.GetMetricName() << std::setw(32) <<
                    metric.GetNamespace();
                const auto &dimensions = metric.GetDimensions();
                for (auto iter = dimensions.cbegin();
                    iter != dimensions.cend(); ++iter)
                {
                    const auto &dimkv = *iter;
                    std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
                    if (iter + 1 != dimensions.cend())
                    {
                        std::cout << ", ";
                    }
                }
                std::cout << std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 ListMetrics。

CLI

AWS CLI

列出 Amazon SNS 的指标

以下 `list-metrics` 示例显示 Amazon SNS 的指标。

```
aws cloudwatch list-metrics \
    --namespace "AWS/SNS"
```

输出：

```
{
    "Metrics": [
```

```
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
```

```
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        }
    ]
}
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 ListMetrics。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListMetrics {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        boolean done = false;
        String nextToken = null;

        try {
            while (!done) {

                ListMetricsResponse response;
                if (nextToken == null) {
                    ListMetricsRequest request = ListMetricsRequest.builder()
                            .namespace(namespace)
                            .build();

                    response = cw.listMetrics(request);
                } else {
                    ListMetricsRequest request = ListMetricsRequest.builder()
                            .namespace(namespace)
                            .nextToken(nextToken)
                            .build();

                    response = cw.listMetrics(request);
                }

                for (Metric metric : response.metrics()) {
                    System.out.printf("Retrieved metric %s",
metric.metricName());
                    System.out.println();
                }

                if (response.nextToken() == null) {
                    done = true;
                } else {
                    nextToken = response.nextToken();
```

```
                }
            }

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 ListMetrics。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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);
};
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 ListMetrics。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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));
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 ListMetrics。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 ListMetrics。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
                       three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {"Namespace": namespace, "MetricName": name}
            if recent:
                kwargs["RecentlyActive"] = "PT3H"  # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
```

```
            raise
        else:
            return metric_iter
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 ListMetrics。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# Lists available metrics for a metric namespace in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric.
# @example
#   list_metrics_for_namespace(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC'
#   )
def list_metrics_for_namespace(cloudwatch_client, metric_namespace)
  response = cloudwatch_client.list_metrics(namespace: metric_namespace)

  if response.metrics.count.positive?
    response.metrics.each do |metric|
      puts "  Metric name: #{metric.metric_name}"
      if metric.dimensions.count.positive?
        puts "    Dimensions:"
        metric.dimensions.each do |dimension|
          puts "      Name: #{dimension.name}, Value: #{dimension.value}"
        end
      else
        puts "No dimensions found."
      end
```

```
    end
  else
    puts "No metrics found for namespace '#{metric_namespace}'. " \
      "Note that it could take up to 15 minutes for recently-added metrics " \
      "to become available."
  end
end

# Example usage:
def run_me
  metric_namespace = "SITE/TRAFFIC"
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = "us-east-1"

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Add three datapoints.
  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisitors",
    "SiteName",
    "example.com",
    5_885.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisits",
    "SiteName",
    "example.com",
    8_628.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "PageViews",
    "PageURL",
    "example.html",
    18_057.0,
```

```
    "Count"
  )

  puts "Metrics for namespace '#{metric_namespace}':"
  list_metrics_for_namespace(cloudwatch_client, metric_namespace)
end

run_me if $PROGRAM_NAME == __FILE__
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 ListMetrics。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   "The following list-metrics example displays the metrics for Amazon
CloudWatch."
   TRY.
       oo_result = lo_cwt->listmetrics(            " oo_result is returned for
testing purposes. "
         iv_namespace = iv_namespace
       ).
       DATA(lt_metrics) = oo_result->get_metrics( ).
       MESSAGE 'Metrics retrieved.' TYPE 'I'.
     CATCH /aws1/cx_cwtinvparamvalueex .
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
   ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 ListMetrics。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **PutAnomalyDetector** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 PutAnomalyDetector。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   /// <summary>
   /// Add an anomaly detector for a single metric.
   /// </summary>
   /// <param name="anomalyDetector">A single metric anomaly detector.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
   {
       var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
           new PutAnomalyDetectorRequest()
           {
               SingleMetricAnomalyDetector = anomalyDetector
           });

       return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
   }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 PutAnomalyDetector。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
   public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .stat("Maximum")
                   .build();

           PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                   .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                   .build();

           cw.putAnomalyDetector(anomalyDetectorRequest);
           System.out.println("Added anomaly detector for metric " +
customMetricName + ".");

       } catch (CloudWatchException | IOException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
```

```
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 PutAnomalyDetector。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 PutAnomalyDetector。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **PutDashboard** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 PutDashboard。

操作示例是大型程序的代码摘录，必须在上下文中运行。在以下代码示例中，您可以查看此操作的上下文：

- 了解基础知识

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string
dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
```

```
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
```

```
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 PutDashboard。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    public static void createDashboardWithMetrics(CloudWatchClient cw, String
dashboardName, String fileName) {
        try {
            PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
                    .dashboardName(dashboardName)
                    .dashboardBody(readFileAsString(fileName))
                    .build();

            PutDashboardResponse response = cw.putDashboard(dashboardRequest);
            System.out.println(dashboardName + " was successfully created.");
            List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
            if (messages.isEmpty()) {
```

```
                System.out.println("There are no messages in the new Dashboard");
            } else {
                for (DashboardValidationMessage message : messages) {
                    System.out.println("Message is: " + message.message());
                }
            }

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 PutDashboard。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
```

```
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 PutDashboard。

PowerShell

适用于 PowerShell 的工具

示例 1：创建或更新名为“Dashboard1”的控制面板，以包含两个并排的指标小部件。

```
$dashBody = @"
{
    "widgets":[
        {
             "type":"metric",
             "x":0,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/EC2",
                      "CPUUtilization",
                      "InstanceId",
                      "i-012345"
                   ]
                ],
                "period":300,
                "stat":"Average",
                "region":"us-east-1",
                "title":"EC2 Instance CPU"
             }
        },
```

```
        {
             "type":"metric",
             "x":12,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/S3",
                      "BucketSizeBytes",
                      "BucketName",
                      "MyBucketName"
                   ]
                ],
                "period":86400,
                "stat":"Maximum",
                "region":"us-east-1",
                "title":"MyBucketName bytes"
            }
        }
    ]
}
"@

Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

示例 2：创建或更新控制面板，将描述控制面板的内容通过管道传输到 cmdlet 中。

```
$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 PutDashboard。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **PutMetricAlarm** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 PutMetricAlarm。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 开始使用告警
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
```

```
            var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
```

```
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 PutMetricAlarm。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

创建告警以观看指标。

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::PutMetricAlarmRequest request;
        request.SetAlarmName(alarm_name);
        request.SetComparisonOperator(
            Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
        request.SetEvaluationPeriods(1);
        request.SetMetricName("CPUUtilization");
        request.SetNamespace("AWS/EC2");
        request.SetPeriod(60);
        request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
        request.SetThreshold(70.0);
        request.SetActionsEnabled(false);
```

```
        request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
        request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("InstanceId");
        dimension.SetValue(instanceId);

        request.AddDimensions(dimension);

        auto outcome = cw.PutMetricAlarm(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to create CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully created CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 PutMetricAlarm。

CLI

AWS CLI

在 CPU 利用率超过 70% 时发送 Amazon Simple Notification Service 电子邮件消息

以下示例使用 `put-metric-alarm` 命令在 CPU 利用率超过 70% 时发送 Amazon Simple Notification Service 电子邮件消息：

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70 percent" --metric-name CPUUtilization --namespace AWS/
EC2 --statistic Average --period 300 --threshold 70 --comparison-
operator GreaterThanThreshold  --dimensions "Name=InstanceId,Value=i-12345678" --
evaluation-periods 2 --alarm-actions arn:aws:sns:us-east-1:111122223333:MyTopic
 --unit Percent
```

如果成功，该命令将返回到提示符。如果已存在同名警报，则该警报将被新警报覆盖。

指定多个维度

以下示例说明了如何指定多个维度。每个维度都指定为一个“名称/值”对，名称和值之间用逗号隔开。多个维度之间用空格隔开：

```
aws cloudwatch put-metric-alarm --alarm-name "Default_Test_Alarm3" --alarm-
description "The default example alarm" --namespace "CW EXAMPLE METRICS"
 --metric-name Default_Test --statistic Average --period 60 --evaluation-
periods 3 --threshold 50 --comparison-operator GreaterThanOrEqualToThreshold --
dimensions Name=key1,Value=value1 Name=key2,Value=value2
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 PutMetricAlarm。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
    public static String createAlarm(CloudWatchClient cw, String fileName) {
        try {
            // Read values from the JSON file.
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();
            String alarmName = rootNode.findValue("exampleAlarmName").asText();
            String emailTopic = rootNode.findValue("emailTopic").asText();
            String accountId = rootNode.findValue("accountId").asText();
            String region = rootNode.findValue("region").asText();

            // Create a List for alarm actions.
            List<String> alarmActions = new ArrayList<>();
            alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
```

```
            PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
                    .alarmActions(alarmActions)
                    .alarmDescription("Example metric alarm")
                    .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
                    .threshold(100.00)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .evaluationPeriods(1)
                    .period(10)
                    .statistic("Maximum")
                    .datapointsToAlarm(1)
                    .treatMissingData("ignore")
                    .build();

            cw.putMetricAlarm(alarmRequest);
            System.out.println(alarmName + " was successfully created!");
            return alarmName;

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 PutMetricAlarm。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 PutMetricAlarm。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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);
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅 AWS SDK for JavaScript API 参考中的 PutMetricAlarm。

Kotlin

SDK for Kotlin

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
suspend fun putMetricAlarm(
    alarmNameVal: String,
    instanceIdVal: String,
) {
    val dimensionOb =
        Dimension {
            name = "InstanceId"
            value = instanceIdVal
        }

    val request =
        PutMetricAlarmRequest {
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanThreshold
            evaluationPeriods = 1
            metricName = "CPUUtilization"
            namespace = "AWS/EC2"
            period = 60
            statistic = Statistic.fromValue("Average")
            threshold = 70.0
```

```
            actionsEnabled = false
            alarmDescription = "An Alarm created by the Kotlin SDK when server
 CPU utilization exceeds 70%"
            unit = StandardUnit.fromValue("Seconds")
            dimensions = listOf(dimensionOb)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created an alarm with name $alarmNameVal")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 PutMetricAlarm。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
```

```
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
```

```
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 PutMetricAlarm。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
# Creates or updates an alarm in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm.
# @param alarm_description [String] A description about the alarm.
# @param metric_name [String] The name of the metric associated with the alarm.
# @param alarm_actions [Array] A list of Strings representing the
#   Amazon Resource Names (ARNs) to execute when the alarm transitions to the
#   ALARM state.
# @param namespace [String] The namespace for the metric to alarm on.
# @param statistic [String] The statistic for the metric.
# @param dimensions [Array] A list of dimensions for the metric, specified as
#   Aws::CloudWatch::Types::Dimension.
# @param period [Integer] The number of seconds before re-evaluating the metric.
# @param unit [String] The unit of measure for the statistic.
```

```
# @param evaluation_periods [Integer] The number of periods over which data is
#   compared to the specified threshold.
# @param theshold [Float] The value against which the specified statistic is
 compared.
# @param comparison_operator [String] The arithmetic operation to use when
#   comparing the specified statistic and threshold.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
# @example
#   exit 1 unless alarm_created_or_updated?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket',
#     'Objects exist in this bucket for more than 1 day.',
#     'NumberOfObjects',
#     ['arn:aws:sns:us-east-1:111111111111:Default_CloudWatch_Alarms_Topic'],
#     'AWS/S3',
#     'Average',
#     [
#       {
#         name: 'BucketName',
#         value: 'doc-example-bucket'
#       },
#       {
#         name: 'StorageType',
#         value: 'AllStorageTypes'
#       }
#     ],
#     86_400,
#     'Count',
#     1,
#     1,
#     'GreaterThanThreshold'
#   )
def alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
```

```
  threshold,
  comparison_operator
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    metric_name: metric_name,
    alarm_actions: alarm_actions,
    namespace: namespace,
    statistic: statistic,
    dimensions: dimensions,
    period: period,
    unit: unit,
    evaluation_periods: evaluation_periods,
    threshold: threshold,
    comparison_operator: comparison_operator
  )
  return true
rescue StandardError => e
  puts "Error creating alarm: #{e.message}"
  return false
end
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 PutMetricAlarm。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
    TRY.
        lo_cwt->putmetricalarm(
          iv_alarmname                = iv_alarm_name
          iv_comparisonoperator       = iv_comparison_operator
          iv_evaluationperiods        = iv_evaluation_periods
          iv_metricname               = iv_metric_name
```

```
        iv_namespace                = iv_namespace
        iv_statistic                = iv_statistic
        iv_threshold                = iv_threshold
        iv_actionsenabled           = iv_actions_enabled
        iv_alarmdescription         = iv_alarm_description
        iv_unit                     = iv_unit
        iv_period                   = iv_period
        it_dimensions               = it_dimensions
      ).
      MESSAGE 'Alarm created.' TYPE 'I'.
    CATCH /aws1/cx_cwtlimitexceededfault.
      MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
  ENDTRY.
```

- 有关 API 详细信息，请参阅 AWS SDK for SAP ABAP API 参考中的 PutMetricAlarm。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

将 **PutMetricData** 与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 PutMetricData。

操作示例是大型程序的代码摘录，必须在上下文中运行。您可以在以下代码示例中查看此操作的上下文：

- 了解基础知识
- 管理指标和告警

.NET

AWS SDK for .NET

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   /// <summary>
   /// Add some metric data using a call to a wrapper class.
   /// </summary>
   /// <param name="customMetricName">The metric name.</param>
   /// <param name="customMetricNamespace">The metric namespace.</param>
   /// <returns></returns>
   private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
       string customMetricNamespace)
   {
       List<MetricDatum> customData = new List<MetricDatum>();
       Random rnd = new Random();

       // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
       var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
       for (int i = 0; i < 10; i++)
       {
           var metricValue = rnd.Next(0, 100);
           customData.Add(
               new MetricDatum
               {
                   MetricName = customMetricName,
                   Value = metricValue,
                   TimestampUtc = utcNowMinus15.AddMinutes(i)
               }
           );
       }

       await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
       return customData;
   }

   /// <summary>
   /// Wrapper to add metric data to a CloudWatch metric.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricData">A data object for the metric data.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutMetricData(string metricNamespace,
       List<MetricDatum> metricData)
```

```
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for .NET API 参考》中的 PutMetricData。

C++

SDK for C++

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

包含所需的文件。

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

将数据放入指标。

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("UNIQUE_PAGES");
        dimension.SetValue("URLS");

        Aws::CloudWatch::Model::MetricDatum datum;
```

```
        datum.SetMetricName("PAGES_VISITED");
        datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
        datum.SetValue(data_point);
        datum.AddDimensions(dimension);

        Aws::CloudWatch::Model::PutMetricDataRequest request;
        request.SetNamespace("SITE/TRAFFIC");
        request.AddMetricData(datum);

        auto outcome = cw.PutMetricData(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to put sample metric data:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully put sample metric data" << std::endl;
        }
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API 参考》中的 PutMetricData。

CLI

AWS CLI

向 Amazon CloudWatch 发布自定义指标

以下示例使用 `put-metric-data` 命令向 Amazon CloudWatch 发布自定义指标：

```
aws cloudwatch put-metric-data --namespace "Usage Metrics" --metric-data file://
metric.json
```

指标本身的值存储在 JSON 文件 `metric.json` 中。

以下是该文件的内容：

```
[
  {
    "MetricName": "New Posts",
    "Timestamp": "Wednesday, June 12, 2013 8:28:20 PM",
```

```
    "Value": 0.50,
    "Unit": "Count"
  }
]
```

有关更多信息，请参阅《Amazon CloudWatch 开发人员指南》中的“发布自定义指标”。

指定多个维度

以下示例说明了如何指定多个维度。每个维度都指定为一个 Name=Value 对。多个维度之间用逗号隔开：

```
aws cloudwatch put-metric-data --metric-name Buffers --
namespace MyNameSpace --unit Bytes --value 231434333 --
dimensions InstanceID=1-23456789,InstanceType=m1.small
```

- 有关 API 详细信息，请参阅《AWS CLI Command Reference》中的 PutMetricData。

Java

SDK for Java 2.x

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

```
   public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
       try {
           // Read values from the JSON file.
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();
```

```
            // Set an Instant object.
            String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
            Instant instant = Instant.parse(time);

            MetricDatum datum = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

            MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

            List<MetricDatum> metricDataList = new ArrayList<>();
            metricDataList.add(datum);
            metricDataList.add(datum2);

            PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

            cw.putMetricData(request);
            System.out.println("Added metric values for for metric " +
customMetricName);

        } catch (CloudWatchException | IOException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的 PutMetricData。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

导入 SDK 和客户端模块，然后调用 API。

```
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();
```

在单独的模块中创建客户端并将其导出。

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 PutMetricData。

SDK for JavaScript (v2)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
// 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));
  }
});
```

- 有关更多信息，请参阅 AWS SDK for JavaScript 开发人员指南。
- 有关 API 详细信息，请参阅《AWS SDK for JavaScript API 参考》中的 PutMetricData。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
 rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
 ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
```

```
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的 PutMetricData。

PowerShell

适用于 PowerShell 的工具

示例 1：创建新的 MetricDatum 对象，并将其写入亚马逊云科技 CloudWatch 指标。

```
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50

### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- 有关 API 详细信息，请参阅《AWS Tools for PowerShell Cmdlet 参考》中的 PutMetricData。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data(self, namespace, name, value, unit):
        """
```

```
        Sends a single data value to CloudWatch for a metric. This metric is
 given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
 name)
            raise
```

将一组数据放入 CloudWatch 指标。

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
```

```
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的 PutMetricData。

Ruby

适用于 Ruby 的 SDK

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
require "aws-sdk-cloudwatch"
```

```
# Adds a datapoint to a metric in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric to add the
#   datapoint to.
# @param metric_name [String] The name of the metric to add the datapoint to.
# @param dimension_name [String] The name of the dimension to add the
#   datapoint to.
# @param dimension_value [String] The value of the dimension to add the
#   datapoint to.
# @param metric_value [Float] The value of the datapoint.
# @param metric_unit [String] The unit of measurement for the datapoint.
# @return [Boolean]
# @example
#   exit 1 unless datapoint_added_to_metric?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC',
#     'UniqueVisitors',
#     'SiteName',
#     'example.com',
#     5_885.0,
#     'Count'
#   )
def datapoint_added_to_metric?(
  cloudwatch_client,
  metric_namespace,
  metric_name,
  dimension_name,
  dimension_value,
  metric_value,
  metric_unit
)
  cloudwatch_client.put_metric_data(
    namespace: metric_namespace,
    metric_data: [
      {
        metric_name: metric_name,
        dimensions: [
          {
            name: dimension_name,
            value: dimension_value
          }
```

```
        ],
        value: metric_value,
        unit: metric_unit
      }
    ]
  )
  puts "Added data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}'."
  return true
rescue StandardError => e
  puts "Error adding data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}': #{e.message}"
  return false
end
```

- 有关 API 详细信息，请参阅《AWS SDK for Ruby API 参考》中的 PutMetricData。

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 的 CloudWatch 场景

以下代码示例向您展示如何通过 AWS SDK 实施 CloudWatch 中的常见场景。这些场景展示了如何通过在 CloudWatch 中调用多个函数或与其他 AWS 服务结合来完成特定任务。每个场景都包含完整源代码的链接，您可以在其中找到有关如何设置和运行代码的说明。

场景以中等水平的经验为目标，可帮助您结合具体环境了解服务操作。

示例

- 通过 AWS SDK 开始使用 CloudWatch 告警
- 使用 AWS SDK 管理 CloudWatch 指标和告警
- 使用 AWS SDK 监控 Amazon DynamoDB 的性能

通过 AWS SDK 开始使用 CloudWatch 告警

以下代码示例展示了如何：

- 创建警报。

- 禁用警报操作。
- 描述警报。
- 删除告警。

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> 查看 GitHub，了解更多信息。查找完整示例，学习如何在 AWS 代码示例存储库中进行设置和运行。

```
DATA lt_alarmnames TYPE /aws1/cl_cwtalarmnames_w=>tt_alarmnames.
DATA lo_alarmname TYPE REF TO /aws1/cl_cwtalarmnames_w.

"Create an alarm"
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                = iv_alarm_name
      iv_comparisonoperator       = iv_comparison_operator
      iv_evaluationperiods        = iv_evaluation_periods
      iv_metricname               = iv_metric_name
      iv_namespace                = iv_namespace
      iv_statistic                = iv_statistic
      iv_threshold                = iv_threshold
      iv_actionsenabled           = iv_actions_enabled
      iv_alarmdescription         = iv_alarm_description
      iv_unit                     = iv_unit
      iv_period                   = iv_period
      it_dimensions               = it_dimensions
    ).
    MESSAGE 'Alarm created' TYPE 'I'.
  CATCH /aws1/cx_cwtlimitexceededfault.
    MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.

"Create an ABAP internal table for the created alarm."
CREATE OBJECT lo_alarmname EXPORTING iv_value = iv_alarm_name.
```

```
    INSERT lo_alarmname INTO TABLE lt_alarmnames.

    "Disable alarm actions."
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames               = lt_alarmnames
        ).
        MESSAGE 'Alarm actions disabled' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_disablealarm_exception).
        DATA(lv_disablealarm_error) = |"{ lo_disablealarm_exception-
>av_err_code }" - { lo_disablealarm_exception->av_err_msg }|.
        MESSAGE lv_disablealarm_error TYPE 'E'.
    ENDTRY.

    "Describe alarm using the same ABAP internal table."
    TRY.
        oo_result = lo_cwt->describealarms(                       " oo_result is
 returned for testing purpose "
          it_alarmnames               = lt_alarmnames
        ).
        MESSAGE 'Alarms retrieved' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_describealarms_exception).
        DATA(lv_describealarms_error) = |"{ lo_describealarms_exception-
>av_err_code }" - { lo_describealarms_exception->av_err_msg }|.
        MESSAGE lv_describealarms_error TYPE 'E'.
    ENDTRY.

    "Delete alarm."
    TRY.
        lo_cwt->deletealarms(
          it_alarmnames = lt_alarmnames
        ).
        MESSAGE 'Alarms deleted' TYPE 'I'.
      CATCH /aws1/cx_cwtresourcenotfound .
        MESSAGE 'Resource being access is not found.' TYPE 'E'.
    ENDTRY.
```

- 有关 API 详细信息，请参阅《AWS SDK for SAP ABAP API 参考》中的以下主题。
  - DeleteAlarms
  - DescribeAlarms
  - DisableAlarmActions

- PutMetricAlarm

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 管理 CloudWatch 指标和告警

以下代码示例展示了如何：

- 创建告警以监视 CloudWatch 指标。
- 将数据放入指标并触发告警。
- 从告警中获取数据。
- 删除告警。

Python

SDK for Python (Boto3)

> **Note**
>
> 查看 GitHub，了解更多信息。在 AWS 代码示例存储库 中查找完整示例，了解如何进行设置和运行。

创建一个包装 CloudWatch 操作的类。

```
from datetime import datetime, timedelta
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
```

```
    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise
```

```
    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
```

```
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm


    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
name)
            raise


    def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
        """
```

```
        Gets statistics for a metric within a specified time span. Metrics are
grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
```

```
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
metric_name)
        return alarm_iter


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

使用包装类将数据放入指标中，触发监控指标的告警，然后从告警中获取数据。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
```

```
    cw_wrapper = CloudWatchWrapper(boto3.resource("cloudwatch"))

    minutes = 20
    metric_namespace = "doc-example-metric"
    metric_name = "page_views"
    start = datetime.utcnow() - timedelta(minutes=minutes)
    print(
        f"Putting data into metric {metric_namespace}.{metric_name} spanning the
"
        f"last {minutes} minutes."
    )
    for offset in range(0, minutes):
        stamp = start + timedelta(minutes=offset)
        cw_wrapper.put_metric_data_set(
            metric_namespace,
            metric_name,
            stamp,
            "Count",
            {
                "values": [
                    random.randint(bound, bound * 2)
                    for bound in range(offset + 1, offset + 11)
                ],
                "counts": [random.randint(1, offset + 1) for _ in range(10)],
            },
        )

    alarm_name = "high_page_views"
    period = 60
    eval_periods = 2
    print(f"Creating alarm {alarm_name} for metric {metric_name}.")
    alarm = cw_wrapper.create_metric_alarm(
        metric_namespace,
        metric_name,
        alarm_name,
        "Maximum",
        period,
        eval_periods,
        100,
        "GreaterThanThreshold",
    )
    print(f"Alarm ARN is {alarm.alarm_arn}.")
    print(f"Current alarm state is: {alarm.state_value}.")
```

```
    print(
        f"Sending data to trigger the alarm. This requires data over the
threshold "
        f"for {eval_periods} periods of {period} seconds each."
    )
    while alarm.state_value == "INSUFFICIENT_DATA":
        print("Sending data for the metric.")
        cw_wrapper.put_metric_data(
            metric_namespace, metric_name, random.randint(100, 200), "Count"
        )
        alarm.load()
        print(f"Current alarm state is: {alarm.state_value}.")
        if alarm.state_value == "INSUFFICIENT_DATA":
            print(f"Waiting for {period} seconds...")
            time.sleep(period)
        else:
            print("Wait for a minute for eventual consistency of metric data.")
            time.sleep(period)
            if alarm.state_value == "OK":
                alarm.load()
                print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Getting data for metric {metric_namespace}.{metric_name} during
timespan "
        f"of {start} to {datetime.utcnow()} (times are UTC)."
    )
    stats = cw_wrapper.get_metric_statistics(
        metric_namespace,
        metric_name,
        start,
        datetime.utcnow(),
        60,
        ["Average", "Minimum", "Maximum"],
    )
    print(
        f"Got {len(stats['Datapoints'])} data points for metric "
        f"{metric_namespace}.{metric_name}."
    )
    pprint(sorted(stats["Datapoints"], key=lambda x: x["Timestamp"]))

    print(f"Getting alarms for metric {metric_name}.")
    alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
```

```
    for alarm in alarms:
        print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

    print(f"Deleting alarms for metric {metric_name}.")
    cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)

    print("Thanks for watching!")
    print("-" * 88)
```

- 有关 API 详细信息，请参阅《AWS SDK for Python (Boto3) API 参考》中的以下主题。
  - DeleteAlarms
  - DescribeAlarmsForMetric
  - DisableAlarmActions
  - EnableAlarmActions
  - GetMetricStatistics
  - ListMetrics
  - PutMetricAlarm
  - PutMetricData

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 监控 Amazon DynamoDB 的性能

以下代码示例显示如何配置应用程序使用 DynamoDB 来监控性能。

Java

SDK for Java 2.x

此示例说明如何配置 Java 应用程序，以监控 DynamoDB 的性能。该应用程序将指标数据发送到 CloudWatch，您可以在其中监控性能。

有关完整的源代码以及如何设置和运行的说明，请参阅 GitHub 上的完整示例。

本示例中使用的服务

- CloudWatch
- DynamoDB

有关 AWS SDK 开发人员指南和代码示例的完整列表，请参阅 将 CloudWatch 与 AWS SDK 结合使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

Amazon CloudWatch 中的安全性

AWS十分重视云安全性。作为AWS客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 - AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS 合规性计划的一部分。要了解适用于 CloudWatch 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性 - 您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Amazon CloudWatch 时应用责任共担模式。它介绍了如何配置 Amazon CloudWatch 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护您的 CloudWatch 资源。

内容

- Amazon CloudWatch 中的数据保护
- 适用于 Amazon CloudWatch 的 Identity and Access Management
- Amazon CloudWatch 的合规性验证
- Amazon CloudWatch 中的恢复能力
- Amazon CloudWatch 中的基础设施安全性
- AWS Security Hub
- 将 CloudWatch 和 CloudWatch Synthetics 与接口 VPC 终端节点结合使用
- Synthetics 金丝雀的安全注意事项

Amazon CloudWatch 中的数据保护

AWS责任共担模式适用于 Amazon CloudWatch 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础架构。您负责维护对托管在此基础架构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management（IAM）设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日记账记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅美国联邦信息处理标准（FIPS）140-3。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您通过控制台、API、AWS CLI 或 AWS 开发工具包使用 CloudWatch 或其他 AWS 服务 时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

传输中加密

CloudWatch 为传输中的数据使用端到端加密。

适用于 Amazon CloudWatch 的 Identity and Access Management

AWS Identity and Access Management（IAM）是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 CloudWatch 资源的人员。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon CloudWatch 如何与 IAM 协同工作
- 适用于 Amazon CloudWatch 的基于身份的策略示例
- Amazon CloudWatch 身份和访问问题排查

- CloudWatch 控制面板权限更新
- 用于 CloudWatch 的 AWS 托管式（预定义）策略
- 客户管理型策略示例
- CloudWatch 对 AWS 托管式策略的更新
- 使用条件键限制对 CloudWatch 命名空间的访问
- 使用条件键限制 Contributor Insights 用户对日志组的访问
- 使用条件键限制告警操作
- 为 CloudWatch 使用服务相关角色
- 对 CloudWatch RUM 使用服务相关角色
- 在 CloudWatch Application Insights 中使用服务相关角色
- 适用于 Amazon CloudWatch Application Insights 的 AWS 托管式策略
- Amazon CloudWatch 权限参考

受众

使用 AWS Identity and Access Management（IAM）的方式因您可以在 CloudWatch 中执行的操作而异。

服务用户 – 如果您使用 CloudWatch 服务来完成工作，则您的管理员会为您提供所需的凭证和权限。当您使用更多 CloudWatch 功能来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 CloudWatch 中的某项功能，请参阅 Amazon CloudWatch 身份和访问问题排查。

服务管理员 - 如果您在公司负责管理 CloudWatch 资源，则您可能具有 CloudWatch 的完全访问权限。您有责任确定您的服务用户应访问哪些 CloudWatch 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 CloudWatch 结合使用的更多信息，请参阅 Amazon CloudWatch 如何与 IAM 协同工作。

IAM 管理员 - 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 CloudWatch 的访问的详细信息。要查看您可在 IAM 中使用的 CloudWatch 基于身份的策略示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

使用身份进行身份验证

身份验证是您使用身份凭证登录 AWS 的方法。您必须作为 AWS 账户根用户、IAM 用户或通过代入 IAM 角色进行身份验证（登录到 AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到 AWS。AWS IAM Identity Center（IAM Identity Center）用户、您公司的单点登录身份验证以及您的 Google 或 Facebook 凭证都是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当您使用联合身份验证访问 AWS 时，您就是在间接代入角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 用户指南》中的如何登录到您的 AWS 账户。

如果您以编程方式访问 AWS，则 AWS 将提供软件开发工具包 (SDK) 和命令行界面 (CLI)，以便使用您的凭证以加密方式签署您的请求。如果您不使用 AWS 工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如，AWS 建议您使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS 中使用多重身份验证（MFA）。

AWS 账户 根用户

当您创建 AWS 账户 时，最初使用的是一个对账户中所有 AWS 服务 和资源拥有完全访问权限的登录身份。此身份称为 AWS 账户 根用户，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）结合使用联合身份验证和身份提供程序，以使用临时凭证来访问 AWS 服务。

联合身份是来自企业用户目录、Web 身份提供程序、AWS Directory Service、Identity Center 目录的用户，或任何使用通过身份源提供的凭证来访问 AWS 服务 的用户。当联合身份访问 AWS 账户 时，他们代入角色，而角色提供临时凭证。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和组，也可以连接并同步到您自己的身份源中的一组用户和组以跨所有 AWS 账户 和应用程序使用。有关 IAM Identity Center 的信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？

IAM 用户和群组

IAM 用户是 AWS 账户 内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

IAM 角色是 AWS 账户 中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。您可以通过切换角色，在 AWS Management Console 中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以担任角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。
- 跨服务访问：某些 AWS 服务使用其它 AWS 服务中的特征。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service (Amazon S3) 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。

- 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务 委派权限的角色。
- 服务相关角色 – 服务相关角色是与 AWS 服务 关联的一种服务角色。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 AWS 账户 中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 Amazon EC2 上运行的应用程序 – 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您将创建策略并将其附加到 AWS 身份或资源，以控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS Management Console、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管式策略是可以附加到 AWS 账户 中的多个用户、组和角色的独立策略。托管式策略包括 AWS 托管式策略和客户管理型策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用来自 IAM 的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Simple Storage Service (Amazon S3)、AWS WAF 和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。

- 服务控制策略 (SCP) – SCP 是 JSON 策略，指定了组织或组织单元 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 服务可以分组和集中管理您的企业拥有的多个 AWS 账户 账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体（包括每个 AWS 账户根用户）的权限。有关 Organizations 和 SCP 的更多信息，请参阅 AWS Organizations 用户指南中的服务控制策略。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅《IAM 用户指南》中的策略评估逻辑。

Amazon CloudWatch 如何与 IAM 协同工作

在使用 IAM 管理对 CloudWatch 的访问之前，您应该了解哪些 IAM 功能可用于 CloudWatch。

下表列出可与 Amazon CloudWatch 搭配使用的 IAM 功能。

IAM 功能	CloudWatch 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键（特定于服务）	是
ACL	否
ABAC（策略中的标签）	部分
临时凭证	是

IAM 功能	CloudWatch 支持
主体权限	是
服务角色	是
服务相关角色	否

要大致了解 CloudWatch 和其他 AWS 服务如何与大多数 IAM 功能结合使用，请参阅《IAM 用户指南》中的与 IAM 结合使用的 AWS 服务。

适用于 CloudWatch 的基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

适用于 CloudWatch 的基于身份的策略示例

要查看适用于 CloudWatch 的基于身份的策略的示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

CloudWatch 内基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户存取，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当主体和资源处于不同的 AWS

账户 中时，则信任账户中的 IAM 管理员还必须授予主体实体（用户或角色）对资源的访问权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

适用于 CloudWatch 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 CloudWatch 操作的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 定义的操作。

CloudWatch 中的策略操作在操作前使用以下前缀：

```
cloudwatch
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "cloudwatch:action1",
      "cloudwatch:action2"
         ]
```

要查看适用于 CloudWatch 的基于身份的策略的示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

适用于 CloudWatch 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN) 指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

有关 CloudWatch 资源类型及其 ARN 的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 Amazon CloudWatch 定义的操作。

要查看适用于 CloudWatch 的基于身份的策略的示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

适用于 CloudWatch 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

有关 CloudWatch 条件键的列表，请参阅《服务授权参考》中的 Amazon CloudWatch 的条件键。要了解您可以对哪些操作和资源使用条件键，请参阅 Amazon CloudWatch 定义的操作。

要查看适用于 CloudWatch 的基于身份的策略的示例，请参阅 适用于 Amazon CloudWatch 的基于身份的策略示例。

CloudWatch 中的 ACL

支持 ACL：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

ABAC 以及 CloudWatch

支持 ABAC（策略中的标签）：部分支持

基于属性的访问控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在 AWS 中，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）以及 AWS 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息,请参阅《IAM 用户指南》中的什么是 ABAC？。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

将临时凭证用于 CloudWatch

支持临时凭证：是

某些 AWS 服务 在您使用临时凭证登录时无法正常工作。有关更多信息，包括 AWS 服务 与临时凭证配合使用，请参阅 IAM 用户指南中的使用 IAM 的 AWS 服务。

如果您不使用用户名和密码而用其它方法登录到 AWS Management Console，则使用临时凭证。例如，当您使用贵公司的单点登录（SSO）链接访问 AWS 时，该过程将自动创建临时凭证。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的切换到角色（控制台）。

您可以使用 AWS CLI 或者 AWS API 创建临时凭证。之后，您可以使用这些临时凭证访问 AWS。AWS 建议您动态生成临时凭证，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

CloudWatch 的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

适用于 CloudWatch 的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务 委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏 CloudWatch 的功能。仅当 CloudWatch 提供相关指导时才编辑服务角色。

适用于 Amazon CloudWatch 的基于身份的策略示例

默认情况下，用户和角色没有创建或修改 CloudWatch 资源的权限。他们也无法使用 AWS Management Console、AWS Command Line Interface（AWS CLI）或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅 IAM 用户指南中的 创建 IAM 策略。

有关 CloudWatch 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 格式，请参阅《服务授权参考》中的 Amazon CloudWatch 的操作、资源和条件键。

主题

- 策略最佳实践
- 使用 CloudWatch 控制台

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 CloudWatch 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- AWS 托管策略及转向最低权限许可入门 – 要开始向用户和工作负载授予权限，请使用 AWS 托管策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。我们建议通过定义特定于您的使用场景的 AWS 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略或工作职能的 AWS 托管策略。
- 应用最低权限 – 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定（AWS 服务例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证（MFA）– 如果您所处的场景要求您的 AWS 账户 中有 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 CloudWatch 控制台

要访问 Amazon CloudWatch 控制台，您必须拥有最低权限。这些权限必须允许您列出和查看有关您 AWS 账户 中的 CloudWatch 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于只需要调用 AWS CLI 或 AWS API 的用户，无需为其提供最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍可使用 CloudWatch 控制台，请同时将 CloudWatch *ConsoleAccess* 或 *ReadOnly* AWS 托管式策略添加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

CloudWatch 控制台所需的权限

下面列出了使用 CloudWatch 控制台所需的完整权限。这些权限提供对 CloudWatch 控制台的完全写入和读取权限。

- application-autoscaling:DescribeScalingPolicies
- autoscaling:DescribeAutoScalingGroups
- autoscaling:DescribePolicies
- cloudtrail:DescribeTrails
- cloudwatch:DeleteAlarms
- cloudwatch:DescribeAlarmHistory
- cloudwatch:DescribeAlarms
- cloudwatch:GetMetricData
- cloudwatch:GetMetricStatistics
- cloudwatch:ListMetrics
- cloudwatch:PutMetricAlarm
- cloudwatch:PutMetricData
- ec2:DescribeInstances
- ec2:DescribeTags
- ec2:DescribeVolumes
- es:DescribeElasticsearchDomain
- es:ListDomainNames
- events:DeleteRule
- events:DescribeRule
- events:DisableRule
- events:EnableRule
- events:ListRules
- events:PutRule

- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStream
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration
- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- logs:CancelExportTask
- logs:CreateExportTask
- logs:CreateLogGroup
- logs:CreateLogStream
- logs:DeleteLogGroup
- logs:DeleteLogStream
- logs:DeleteMetricFilter
- logs:DeleteRetentionPolicy
- logs:DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries

- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter
- s3:CreateBucket
- s3:ListBucket
- sns:CreateTopic
- sns:GetTopicAttributes
- sns:ListSubscriptions
- sns:ListTopics
- sns:SetTopicAttributes
- sns:Subscribe
- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs:ListQueues
- sqs:SetQueueAttributes
- swf:CreateAction
- swf:DescribeAction
- swf:ListActionTemplates
- swf:RegisterAction
- swf:RegisterDomain

- swf:UpdateAction

此外，要查看 X-Ray 跟踪地图，您需要 AWSXrayReadOnlyAccess

Amazon CloudWatch 身份和访问问题排查

使用以下信息可帮助您诊断和修复在使用 CloudWatch 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 CloudWatch 中执行操作
- 我无权执行 iam:PassRole
- 我希望允许我的 AWS 账户 以外的人访问我的 CloudWatch 资源

我无权在 CloudWatch 中执行操作

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 cloudwatch:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cloudwatch:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 cloudwatch:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam:PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略才能将角色传递给 CloudWatch。

有些 AWS 服务 允许将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 CloudWatch 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系 AWS 管理员。您的管理员是提供登录凭证的人。

我希望允许我的 AWS 账户 以外的人访问我的 CloudWatch 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 CloudWatch 是否支持这些功能，请参阅 Amazon CloudWatch 如何与 IAM 协同工作。
- 要了解如何为您拥有的 AWS 账户中的资源提供访问权限，请参阅《IAM 用户指南》中的为您拥有的另一个 AWS 账户中的 IAM 用户提供访问权限。
- 要了解如何为第三方 AWS 账户 提供您的资源的访问权限，请参阅 IAM 用户指南中的为第三方拥有的 AWS 账户 提供访问权限。
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

CloudWatch 控制面板权限更新

从 2018 年 5 月 1 日起，AWS 改变了访问 CloudWatch 控制面板所需的权限。现在，CloudWatch 控制台中的控制面板访问需要在 2017 年推出的权限，以支持控制面板 API 操作：

- cloudwatch:GetDashboard
- cloudwatch:ListDashboards
- cloudwatch:PutDashboard
- cloudwatch:DeleteDashboards

要访问 CloudWatch 控制面板，您需要以下项目之一：

- AdministratorAccess 策略。
- CloudWatchFullAccess 策略。
- 包含以下一个或多个特定权限的自定义策略：
  - `cloudwatch:GetDashboard` 和 `cloudwatch:ListDashboards` 能够查看控制面板
  - `cloudwatch:PutDashboard` 能够创建或修改控制面板
  - `cloudwatch:DeleteDashboards` 能够删除控制面板

有关使用策略更改 IAM 用户权限的更多信息，请参阅更改 IAM 用户的权限。

有关 CloudWatch 权限的更多信息，请参阅 Amazon CloudWatch 权限参考。

有关控制面板 API 操作的更多信息，请参阅《Amazon CloudWatch API 参考》中的 PutDashboard。

用于 CloudWatch 的 AWS 托管式（预定义）策略

AWS 通过提供由 AWS 创建和管理的独立 IAM 策略来满足许多常用案例的要求。这些 AWS 托管策略可针对常用案例授予必要的权限，使您免去调查所需权限的工作。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

以下 AWS 托管式策略（可附加到您账户中的用户）特定于 CloudWatch。

主题

- CloudWatchFullAccessV2
- CloudWatchFullAccess
- CloudWatchReadOnlyAccess
- CloudWatchActionsEC2Access
- CloudWatchAutomaticDashboardsAccess
- CloudWatchAgentServerPolicy
- CloudWatchAgentAdminPolicy
- 用于 CloudWatch 跨账户可观测性的 AWS 托管（预定义）策略
- 用于 CloudWatch Application Signals 的 AWS 托管（预定义）策略
- 用于 CloudWatch Synthetics 的 AWS 托管式（预定义）策略
- 适用于 Amazon CloudWatch RUM 的 AWS 托管式（预定义）策略
- 适用于 CloudWatch Evidently 的 AWS 托管式（预定义）策略
- 适用于 AWS Systems Manager Incident Manager 的 AWS 托管式策略

CloudWatchFullAccessV2

AWS 最近添加了 CloudWatchFullAccessV2 托管 IAM 策略。此策略授予对 CloudWatch 操作和资源的完全访问权限，并更正确地确定授予其他服务（如 Amazon SNS 和 Amazon EC2 Auto Scaling）的权限范围。我们建议您开始使用此策略，而不是使用 CloudWatchFullAccess。AWS 计划在不久的将来弃用 CloudWatchFullAccess 。

它包括 `application-signals:` 权限，以便用户可以从 CloudWatch 控制台的 Application Signals 下访问所有功能。它包含一些 `autoscaling:Describe` 权限，以便使用此策略的用户可以查看与 CloudWatch 警报关联的自动扩缩操作。它包含一些 `sns` 权限，以便使用此策略的用户可以检索、创建 Amazon SNS 主题并将其与 CloudWatch 警报关联。它包含 IAM 权限，以便使用此策略的用户可以查看有关与 CloudWatch 关联的服务相关角色的信息。它包含 `oam:ListSinks` 和 `oam:ListAttachedLinks` 权限，以便使用此策略的用户可以借助控制台在 CloudWatch 跨账户可观察性中查看源账户共享的数据。

其中包括 `rum`、`synthetics` 和 `xray` 权限，因此用户可以完全访问 CloudWatch Synthetics、AWS X-Ray 和 CloudWatch RUM，所有这些都在 CloudWatch 服务范围之内。

CloudWatchFullAccessV2 的内容如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchFullAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
                "application-signals:*",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribePolicies",
                "cloudwatch:*",
                "logs:*",
                "sns:CreateTopic",
                "sns:ListSubscriptions",
                "sns:ListSubscriptionsByTopic",
                "sns:ListTopics",
                "sns:Subscribe",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks",
```

```
                "rum:*",
                "synthetics:*",
                "xray:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsServiceLinkedRolePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "application-
signals.cloudwatch.amazonaws.com"
                }
            }
        },
        {
            "Sid": "EventsServicePermissions",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        },
         {
            "Sid": "OAMReadPermissions",
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }
    ]
}
```

CloudWatchFullAccess

CloudWatchFullAccess 策略即将被弃用。我们建议您停止使用它，改用 CloudWatchFullAccessV2。

CloudWatchFullAccess 的内容如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:*",
                "logs:*",
                "sns:*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "oam:ListSinks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }
    ]
}
```

CloudWatchReadOnlyAccess

CloudWatchReadOnlyAccess 策略授予对 CloudWatch 的只读访问权限。

策略包含一些 `logs:` 权限，因此拥有此策略的用户可以使用控制台查看 CloudWatch 日志信息和 CloudWatch Logs Insights 查询。其中包含 `autoscaling:Describe*`，因此拥有此策略的用户可以查看与 CloudWatch 警报关联的 Auto Scaling 操作。它包括 `application-signals:` 权限，以便用户可以使用 Application Signals 来监控其服务的运行状况。其中包含 `application-autoscaling:DescribeScalingPolicies`，因此拥有此策略的用户可以访问有关 Application Auto Scaling 策略的信息。其中包含 `sns:Get*` 和 `sns:List*`，因此拥有此策略的用户可以检索有关接收 CloudWatch 警报通知的 Amazon SNS 主题的信息。其中包含 `oam:ListSinks` 和 `oam:ListAttachedLinks` 权限，因此拥有此策略的用户可以使用控制台在 CloudWatch 跨账户可观测性中查看源账户共享的数据。它包括 `iam:GetRole` 权限，以便用户可以检查是否已设置 CloudWatch Application Signals。

其中包含 `rum`、`synthetics` 和 `xray` 权限，因此用户可以只读访问 CloudWatch Synthetics、AWS X-Ray 和 CloudWatch RUM，所有这些都在 CloudWatch 服务范围之内。

CloudWatchReadOnlyAccess 策略的内容如下。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchReadOnlyAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalingPolicies",
                "application-signals:BatchGet*",
                "application-signals:Get*",
                "application-signals:List*",
                "autoscaling:Describe*",
                "cloudwatch:BatchGet*",
                "cloudwatch:Describe*",
                "cloudwatch:GenerateQuery",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
```

```
                "logs:Describe*",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents",
                "logs:StartLiveTail",
                "logs:StopLiveTail",
                "oam:ListSinks",
                "sns:Get*",
                "sns:List*",
                "rum:BatchGet*",
                "rum:Get*",
                "rum:List*",
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "xray:BatchGet*",
                "xray:Get*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "OAMReadPermissions",
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        },
        {
            "Sid": "CloudWatchReadOnlyGetRolePermissions",
            "Effect": "Allow",
            "Action": "iam:GetRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
        }
    ]
}
```

CloudWatchActionsEC2Access

CloudWatchActionsEC2Access 策略授予对 CloudWatch 告警和指标，以及 Amazon EC2 元数据的只读访问权限。其还授予对 EC2 实例的停止、终止和重启 API 操作的访问权限。

CloudWatchActionsEC2Access 策略的内容如下。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Describe*",
        "ec2:Describe*",
        "ec2:RebootInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

CloudWatchAutomaticDashboardsAccess

CloudWatch-CrossAccountAccess 托管式策略由CloudWatch-CrossAccountSharingRole IAM 角色使用。此角色和策略使跨账户控制面板的用户能够查看共享仪表板的各个账户中的自动控制面板。

CloudWatchAutomaticDashboardsAccess 策略的内容如下所示：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "cloudfront:GetDistribution",
        "cloudfront:ListDistributions",
        "dynamodb:DescribeTable",
        "dynamodb:ListTables",
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "elasticache:DescribeCacheClusters",
        "elasticbeanstalk:DescribeEnvironments",
```

```
        "elasticfilesystem:DescribeFileSystems",
        "elasticloadbalancing:DescribeLoadBalancers",
        "kinesis:DescribeStream",
        "kinesis:ListStreams",
        "lambda:GetFunction",
        "lambda:ListFunctions",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "resource-groups:ListGroupResources",
        "resource-groups:ListGroups",
        "route53:GetHealthCheck",
        "route53:ListHealthChecks",
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "sns:ListTopics",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "sqs:ListQueues",
        "synthetics:DescribeCanariesLastRun",
        "tag:GetResources"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "apigateway:GET"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:apigateway:*::/restapis*"
      ]
    }
  ]
```

CloudWatchAgentServerPolicy

CloudWatchAgentServerPolicy 策略可用于附加到 Amazon EC2 实例的 IAM 角色中，以允许 CloudWatch 代理从实例读取信息并将其写入 CloudWatch。其包含以下内容。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "CWACloudWatchServerPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeVolumes",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CWASSMServerPermissions",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
        }
    ]
}
```

CloudWatchAgentAdminPolicy

CloudWatchAgentAdminPolicy 策略可用于附加到 Amazon EC2 实例的 IAM 角色。此策略允许 CloudWatch 代理从实例读取信息并将其写入 CloudWatch，还可以将信息写入 Parameter Store。其包含以下内容。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CWACloudWatchPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ec2:DescribeTags",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CWASSMPermissions",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
        }
    ]
}
```

> **Note**
>
> 您可以通过登录到 IAM 控制台并在该控制台中搜索特定策略来查看这些权限策略。

此外，您还可以创建您自己的自定义 IAM 策略，以授予对 CloudWatch 操作和资源的相关权限。您可以将这些自定义策略附加到需要这些权限的 IAM 用户或组。

用于 CloudWatch 跨账户可观测性的 AWS 托管（预定义）策略

本节中的策略授予与 CloudWatch 跨账户可观测性相关的权限。有关更多信息，请参阅 CloudWatch 跨账户可观测性。

CloudWatchCrossAccountSharingConfiguration

CloudWatchCrossAccountSharingConfiguration 策略授予可创建、管理和查看可观测性访问管理器链接的权限，用于在账户之间共享 CloudWatch 资源。有关更多信息，请参阅 CloudWatch 跨账户可观测性。内容如下：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:Link",
                "oam:ListLinks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        }
    ]
}
```

OAMFullAccess

OAMFullAccess 策略授予可创建、管理和查看可观测性访问管理器汇点和链接的权限，这些汇点和链接用于 CloudWatch 跨账户可观测性。

OAMFullAccess 策略本身不允许您跨链接共享可观测性数据。要创建可共享 CloudWatch 指标的链接，您还需要 CloudWatchFullAccess 或 CloudWatchCrossAccountSharingConfiguration。要创建可共享 CloudWatch Logs 日志组的链接，您还需要 CloudWatchLogsFullAccess 或 CloudWatchLogsCrossAccountSharingConfiguration。要创建可共享 X-Ray 追踪信息的链接，您还需要 AWSXRayFullAccess 或 AWSXRayCrossAccountSharingConfiguration。

有关更多信息，请参阅 CloudWatch 跨账户可观测性。内容如下：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:*"
            ],
            "Resource": "*"
        }
    ]
}
```

OAMReadOnlyAccess

OAMReadOnlyAccess 策略授予 Observability Access Manager 资源的只读访问权限，用于 CloudWatch 跨账户可观测性。有关更多信息，请参阅 CloudWatch 跨账户可观测性。内容如下：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "oam:Get*",
                "oam:List*"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

用于 CloudWatch Application Signals 的 AWS 托管（预定义）策略

本节中的策略授予与 CloudWatch Application Signals 相关的权限。有关更多信息，请参阅 Application Signals。

CloudWatchApplicationSignalsReadOnlyAccess

AWS 添加了 CloudWatchApplicationSignalsReadOnlyAccess 托管 IAM 策略。此策略授予用户对 CloudWatch 控制台中 Application Signals 下可用操作和资源的只读访问权限。它包括 `application-signals:` 策略，以使用户可以使用 CloudWatch Application Signals 来查看、调查和监控其服务的运行状况。它包括一项允许用户检索 IAM 角色相关信息的 `iam:GetRole` 策略。它包括启动和停止查询、检索指标筛选器的配置以及获取查询结果的 `logs:` 策略。它包括 `cloudwatch:` 策略，以使用户能够获取有关 CloudWatch 警报或指标的信息。它包括 `synthetics:` 策略，以使用户能够检索有关 Synthetics canary 运行的信息。它包括运行批量操作、检索数据和更新 RUM 客户端指标定义的 `rum:` 策略。它包括一项检索跟踪摘要的 `xray:` 策略。

CloudWatchApplicationSignalsReadOnlyAccess 策略的内容如下所示。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
   "Effect": "Allow",
   "Action": [
    "application-signals:BatchGetServiceLevelObjectiveBudgetReport",
    "application-signals:GetService",
    "application-signals:GetServiceLevelObjective",
    "application-signals:ListServiceLevelObjectives",
    "application-signals:ListServiceDependencies",
    "application-signals:ListServiceDependents",
    "application-signals:ListServiceOperations",
    "application-signals:ListServices",
    "application-signals:ListTagsForResource"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
   "Effect": "Allow",
```

```
    "Action": "iam:GetRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
   },
   {
    "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
    "Effect": "Allow",
    "Action": [
     "logs:StartQuery"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsLogsPermissions",
    "Effect": "Allow",
    "Action": [
     "logs:StopQuery",
     "logs:GetQueryResults"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "cloudwatch:DescribeAlarms"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "cloudwatch:GetMetricData",
     "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
   },
   {
    "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
    "Effect": "Allow",
    "Action": [
     "synthetics:DescribeCanaries",
     "synthetics:DescribeCanariesLastRun",
```

```
    "synthetics:GetCanaryRuns"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
   "Effect": "Allow",
   "Action": [
    "rum:BatchGetRumMetricDefinitions",
    "rum:GetAppMonitor",
    "rum:GetAppMonitorData",
    "rum:ListAppMonitors"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
   "Effect": "Allow",
   "Action": [
    "xray:GetTraceSummaries"
   ],
   "Resource": "*"
  }
 ]
}
```

CloudWatchApplicationSignalsFullAccess

AWS 添加了 CloudWatchApplicationSignalsFullAccess 托管 IAM 策略。此策略授予用户对 CloudWatch 控制台中所有可用操作和资源的访问权限。它包括 `application-signals:` 策略，以使用户可以使用 CloudWatch Application Signals 来查看、调查和监控其服务的运行状况。它使用 `cloudwatch:` 策略从指标和警报中检索数据。它使用 `logs:` 策略来管理查询和筛选器。它使用 `synthetics:` 策略，以使用户能够检索有关 Synthetics canary 运行的信息。它包括运行批量操作、检索数据和更新 RUM 客户端指标定义的 `rum:` 策略。它包括一项检索跟踪摘要的 `xray:` 策略。它包括 `arn:aws:cloudwatch:*:*:alarm:` 策略，以使用户能够检索有关服务级别目标（SLO）警报的信息。它包括管理 IAM 角色的 `iam:` 策略。它使用 `sns:` 策略创建、列出和订阅 Amazon SNS 主题。

CloudWatchApplicationSignalsFullAccess 策略的内容如下所示。

```
{
 "Version": "2012-10-17",
 "Statement": [
```

```
    {
        "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
        "Effect": "Allow",
        "Action": "application-signals:*",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:DescribeAlarms",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StartQuery"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsLogsPermissions",
        "Effect": "Allow",
        "Action": [
            "logs:StopQuery",
            "logs:GetQueryResults"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
        "Effect": "Allow",
        "Action": [
            "synthetics:DescribeCanaries",
            "synthetics:DescribeCanariesLastRun",
```

```
            "synthetics:GetCanaryRuns"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsRumPermissions",
        "Effect": "Allow",
        "Action": [
            "rum:BatchCreateRumMetricDefinitions",
            "rum:BatchDeleteRumMetricDefinitions",
            "rum:BatchGetRumMetricDefinitions",
            "rum:GetAppMonitor",
            "rum:GetAppMonitorData",
            "rum:ListAppMonitors",
            "rum:PutRumMetricsDestination",
            "rum:UpdateRumMetricDefinition"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsXrayPermissions",
        "Effect": "Allow",
        "Action": "xray:GetTraceSummaries",
        "Resource": "*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
        "Effect": "Allow",
        "Action": "cloudwatch:PutMetricAlarm",
        "Resource": [
            "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
            "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
        ]
    },
    {
        "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
        "Effect": "Allow",
        "Action": "iam:CreateServiceLinkedRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
        "Condition": {
            "StringLike": {
                "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
```

```
            }
        }
    },
    {
        "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
        "Effect": "Allow",
        "Action": "iam:GetRole",
        "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:Subscribe"
        ],
        "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
    },
    {
        "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
        "Effect": "Allow",
        "Action": "sns:ListTopics",
        "Resource": "*"
    }
 ]
}
```

用于 CloudWatch Synthetics 的 AWS 托管式（预定义）策略

CloudWatchSyntheticsFullAccess 和 CloudWatchSyntheticsReadOnlyAccess AWS 托管式策略可供您分配给将要管理或使用 CloudWatch Synthetics 的用户。以下其他策略也是相关的：

- AmazonS3ReadOnlyAccess 和 CloudWatchReadOnlyAccess – 在 CloudWatch 控制台中读取所有 Synthetics 数据所必需的策略。
- AWSLambdaReadOnlyAccess – 可查看 Canary 使用的源代码。
- CloudWatchSyntheticsFullAccess 使您能够创建 Canary。此外，要创建和删除将为其创建新 IAM 角色的 Canary，您还需要以下内联策略语句：

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

Important

授予用户 `iam:CreateRole`、`iam:DeleteRole`、`iam:CreatePolicy`、`iam:DeletePolicy`、`iam:Atta` 和 `iam:DetachRolePolicy` 权限，将授予用户完全管理访问权限，用户可以创建、附加和删除具有匹配 `arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*` 和 `arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*` 的 ARN 的角色和策略。例如，拥有这些权限的用户可以创建一个对所有资源具有完全权限的策略，并将该策略附加到匹配该 ARN 模式的任何角色。请谨慎地为相关人员授予这些权限。

有关附加策略和向用户授予权限的信息，请参阅更改 IAM 用户的权限和为用户或角色嵌入内联策略。

CloudWatchSyntheticsFullAccess

CloudWatchSyntheticsFullAccess 策略的内容如下。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::cw-syn-*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::aws-synthetics-library-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "synthetics.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:PublishVersion",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunctionConfiguration",
                "lambda:DeleteFunction"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:function:cwsyn-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:layer:cwsyn-*",
                "arn:aws:lambda:*:*:layer:Synthetics:*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "s3.*.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

CloudWatchSyntheticsReadOnlyAccess

CloudWatchSyntheticsReadOnlyAccess 策略的内容如下。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

适用于 Amazon CloudWatch RUM 的 AWS 托管式（预定义）策略

您可以将 AWS 托管式策略 AmazonCloudWatchRUMFullAccess 和 AmazonCloudWatchRUMReadOnlyAccess 分配给将管理或使用 CloudWatch RUM 的用户。

AmazonCloudWatchRUMFullAccess

AmazonCloudWatchRUMFullAccess 策略的内容如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/rum.amazonaws.com/
AWSServiceRoleForRealUserMonitoring"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/RUM-Monitor*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "cognito-identity.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:alarm:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:CreateIdentityPool",
                "cognito-identity:ListIdentityPools",
                "cognito-identity:DescribeIdentityPool",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:SetIdentityPoolRoles"
            ],
            "Resource": "arn:aws:cognito-identity:*:*:identitypool/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DeleteLogGroup",
                "logs:PutRetentionPolicy",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:*RUMService*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:DescribeResourcePolicies"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "arn:aws:logs:*:*:log-group::log-stream:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:describeCanaries",
                "synthetics:describeCanariesLastRun"
            ],
            "Resource": "arn:aws:synthetics:*:*:canary:*"
        }
    ]
}
```

AmazonCloudWatchRUMReadOnlyAccess

AmazonCloudWatchRUMReadOnlyAccess 策略的内容如下所示。

```
{

 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "rum:GetAppMonitor",
    "rum:GetAppMonitorData",
    "rum:ListAppMonitors",
    "rum:ListRumMetricsDestinations",
    "rum:BatchGetRumMetricDefinitions"
   ],
   "Resource": "*"
  }
 ]
}
```

AmazonCloudWatchRUMServiceRolePolicy

您无法将 AmazonCloudWatchRUMServiceRolePolicy 附加到 IAM 实体。此策略会附加到允许 CloudWatch RUM 向其他相关 AWS 服务发布监控数据的服务相关角色。有关此服务相关角色的更多信息，请参阅 对 CloudWatch RUM 使用服务相关角色。

AmazonCloudWatchRUMServiceRolePolicy 的完整内容如下所示。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

适用于 CloudWatch Evidently 的 AWS 托管式（预定义）策略

您可以将 AWS 托管式策略 CloudWatchSyntheticsFullAccess 和 CloudWatchSyntheticsReadOnlyAccess 分配给将管理或使用 CloudWatch Evidently 的用户。

CloudWatchEvidentlyFullAccess

CloudWatchEvidentlyFullAccess 策略的内容如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarmHistory",
```

```
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:ListTagsForResource"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:TagResource",
                "cloudwatch:UnTagResource"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudtrail:LookupEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Evidently-Alarm-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Evidently-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

CloudWatchEvidentlyReadOnlyAccess

CloudWatchEvidentlyReadOnlyAccess 策略的内容如下所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "evidently:GetExperiment",
                "evidently:GetFeature",
                "evidently:GetLaunch",
                "evidently:GetProject",
                "evidently:GetSegment",
                "evidently:ListExperiments",
                "evidently:ListFeatures",
                "evidently:ListLaunches",
                "evidently:ListProjects",
                "evidently:ListSegments",
                "evidently:ListSegmentReferencs"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

适用于 AWS Systems Manager Incident Manager 的 AWS 托管式策略

AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy 策略附加到一个与服务相关的角色，该角色允许 CloudWatch 在 AWS Systems Manager Incident Manager 中代表您启动事件。有关更多信息，请参阅 CloudWatch 告警 Systems Manager Incident Manager 操作的服务相关角色权限。

该策略具有以下权限：

- ssm-incidents:StartIncident

客户管理型策略示例

本节的用户策略示例介绍如何授予对各 CloudWatch 操作的权限。当您使用 CloudWatch API、AWS SDK 或 AWS CLI 时，可以使用这些策略。

示例

- 示例 1：允许用户对 CloudWatch 进行完全访问
- 示例 2：允许对 CloudWatch 进行只读访问
- 示例 3：停止或终止 Amazon EC2 实例

示例 1：允许用户对 CloudWatch 进行完全访问

要授予用户对 CloudWatch 的完全访问权限，您可以使用授予用户 CloudWatchFullAccess 托管式策略，而不必创建客户托管式策略。CloudWatchFullAccess 策略的内容列在 CloudWatchFullAccess 中。

示例 2：允许对 CloudWatch 进行只读访问

以下策略允许用户对 CloudWatch 进行只读访问以及查看 Amazon EC2 Auto Scaling 操作、CloudWatch 指标、CloudWatch Logs 数据以及告警相关 Amazon SNS 数据。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
        "logs:Get*",
        "logs:Describe*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "sns:Get*",
        "sns:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

示例 3：停止或终止 Amazon EC2 实例

以下策略允许 CloudWatch 告警操作停止或终止 EC2 实例。在以下示例中，GetMetricData、ListMetrics 和 DescribeAlarms 操作是可选的。建议您选择这些操作以确保正确停止或终止了实例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

CloudWatch 对 AWS 托管式策略的更新

查看有关 CloudWatch 的 AWS 托管式策略更新的详细信息（从该服务开始跟踪这些更改开始）。有关此页面更改的自动提示，请订阅 CloudWatch 文档历史记录页面上的 RSS 源。

更改	描述	日期
CloudWatchApplicationSignalsReadOnlyAccess – 新策略	CloudWatch 创建了一项名为 CloudWatchApplicationSignalsReadOnlyAccess 的新策略。 此策略授予对 Application Signals 的 CloudWatch 控制台中可用资源和操作的只读访问权限。此策略的范围包括 `application-signals:` 策略，以使用户能够使用 CloudWatch 控制台中 Application Signals 下可用的只读操作和资源。它包含管理 IAM 角色的 `iam:` 策略。它包含管理日志查询和筛选器的一些 `logs:` 策略。它包含检索 CloudWatch 警报和指标	2024 年 6 月 7 日

更改	描述	日期
	相关信息的 `cloudwatch:` 策略。它包含检索 Synthetics Canary 相关信息的一些 `synthetics:` 策略。它包含管理 RUM 客户端和作业的 `rum:` 策略。它包含获取跟踪摘要的 `xray:` 策略。	

更改	描述	日期
CloudWatchApplicationSignalsFullAccess – 新策略	CloudWatch 创建了一项名为 CloudWatchApplicationSignalsFullAccess 的新策略。 此策略授予对 Application Signals 的 CloudWatch 控制台中可用资源和操作的完全访问权限。此策略的范围包括 `application-signals:` ，以使用户能够使用 Application Signals 操作和资源。它包含检索 CloudWatch 警报和指标相关信息的一些 `cloudwatch:` 策略。它包含管理日志查询的一些 `logs:` 策略。它包含写入和读取 Synthetics Canary 相关信息的一些 `synthetics:` 策略。它包含管理 RUM 客户端和作业的 `rum:` 策略。它包含获取跟踪摘要的 `xray:` 策略。它包含管理 CloudWatch 警报的一些 `cloudwatch:` 策略。它包含管理 IAM 角色的一些 `iam:` 策略。它包含管理 Amazon Simple Notification Service 通知的一些 `sns:` 策略。	2024 年 6 月 7 日

更改	描述	日期
CloudWatchFullAccessV2：对现有策略的更新	CloudWatch 更新了名为 CloudWatchFullAccessV2 的策略。 `CloudWatchFullAccessPermissions` 策略的范围已更新为添加 `application-signals:*` ，以便用户可以使用 CloudWatch Application Signals 来查看、调查和诊断其服务运行状况的问题。	2024 年 5 月 20 日

更改	描述	日期
CloudWatchReadOnlyAccess – 对现有策略的更新	CloudWatch 更新了名为 CloudWatchReadOnlyAccess 的策略。 `CloudWatchReadOnlyAccessPermissions` 策略的范围已更新为添加 `application-signals:BatchGet*` 、`application-signals:List*` 和 `application-signals:Get*` ，以便用户可以使用 CloudWatch Application Signals 来查看、调查和诊断其服务运行状况的问题。`CloudWatchReadOnlyGetRolePermissions` 的范围已更新为添加 `iam:GetRole` 操作，以便用户可以检查是否已设置 CloudWatch Application Signals。	2024 年 5 月 20 日

更改	描述	日期
CloudWatchApplicationSignalsServiceRolePolicy – 更新为现有策略	CloudWatch 更新了名为 CloudWatchApplicationSignalsServiceRolePolicy 的策略。 `logs:StartQuery` 和 `logs:GetQueryResults` 权限的范围已更改，添加 `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` 和 `arn:aws:logs:*:*:log-group:/aws/application-signals/data:*` ARN 以在更多架构上启用 Application Signals。	2024 年 4 月 18 日
CloudWatchApplicationSignalsServiceRolePolicy – 更新为现有策略	CloudWatch 更改了 CloudWatchApplicationSignalsServiceRolePolicy 中权限的范围。 `cloudwatch:GetMetricData` 权限的范围已更改为 `*`，以使 Application Signals 可以从关联账户中的源检索指标。	2024 年 4 月 8 日

更改	描述	日期
CloudWatchAgentServerPolicy – 更新到现有策略	CloudWatch 已将权限添加到 CloudWatchAgentServerPolicy 。 已添加 `xray:PutTraceSegments` 、`xray:PutTelemetryRecords` 、`xray:GetSamplingRules` 、`xray:GetSamplingTargets` 、`xray:GetSamplingStatisticSummaries` 和 `logs:PutRetentionPolicy` 权限，从而使 CloudWatch 代理可以发布 X-Ray 跟踪和修改日志组保留期。	2024 年 2 月 12 日

更改	描述	日期
CloudWatchAgentAdminPolicy – 更新到现有策略	CloudWatch 已将权限添加到 CloudWatchAgentAdminPolicy。 已添加 `xray:PutTraceSegments` 、`xray:PutTelemetryRecords` 、`xray:GetSamplingRules` 、`xray:GetSamplingTargets` 、`xray:GetSamplingStatisticSummaries` 和 `logs:PutRetentionPolicy` 权限，从而使 CloudWatch 代理可以发布 X-Ray 跟踪和修改日志组保留期。	2024 年 2 月 12 日

更改	描述	日期
CloudWatchFullAccessV2：对现有策略的更新	CloudWatch 添加了对 CloudWatchFullAccessV2 的权限。 添加了 CloudWatch Synthetics、X-Ray 和 CloudWatch RUM 操作的现有权限以及 CloudWatch Application Signals 的新权限，因此拥有此策略的用户可以管理 CloudWatch Application Signals。 添加了创建 CloudWatch Application Signals 服务相关角色的权限，以允许 CloudWatch Application Signals 发现日志、指标、跟踪和标签中的遥测数据。	2023 年 12 月 5 日

更改	描述	日期
CloudWatchReadOnlyAccess – 对现有策略的更新	CloudWatch 添加了对 CloudWatchReadOnlyAccess 的权限。 添加了 CloudWatch Synthetics、X-Ray 和 CloudWatch RUM 操作的现有只读权限以及 CloudWatch Application Signals 的新只读权限，因此拥有此策略的用户可以对 CloudWatch Application Signals 报告的服务运行状况问题进行分类和诊断。 添加了 `cloudwatch:GenerateQuery` 权限，以便拥有此策略的用户可以根据自然语言提示来生成 CloudWatch Metrics Insights 查询字符串。	2023 年 12 月 5 日
CloudWatchReadOnlyAccess – 对现有策略的更新。	CloudWatch 已向 CloudWatchReadOnlyAccess 添加了权限。 添加了 `cloudwatch:GenerateQuery` 权限，以便拥有此策略的用户可以根据自然语言提示来生成 CloudWatch Metrics Insights 查询字符串。	2023 年 12 月 1 日

更改	描述	日期
CloudWatchApplicationSignalsServiceRolePolicy – 新策略	CloudWatch 添加了一项新策略 CloudWatchApplicationSignalsServiceRolePolicy。 CloudWatchApplicationSignalsServiceRolePolicy 会授予一项即将推出的功能权限，用于收集 CloudWatch Logs 数据、X-Ray 跟踪数据、CloudWatch 指标数据和标记数据。	2023 年 11 月 9 日
AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy – 新策略	CloudWatch 添加了一个新策略 AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy。 AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy 授予 CloudWatch 代表您从数据库获取性能详情指标的权限。	2023 年 9 月 20 日
CloudWatchReadOnlyAccess – 对现有策略的更新	CloudWatch 已向 CloudWatchReadOnlyAccess 添加了权限。 添加了 `application-autoscaling:DescribeScalingPolicies` 权限，以便拥有此策略的用户可以访问有关 Application Auto Scaling 策略的信息。	2023 年 9 月 14 日

更改	描述	日期
CloudWatchFullAccessV2 – 新策略	CloudWatch 添加了一项新策略 CloudWatchFullAccessV2。 CloudWatchFullAccessV2 授予对 CloudWatch 操作和资源的完全访问权限，同时可以更好地限定授予其他服务（例如 Amazon SNS 和 Amazon EC2 Auto Scaling）的权限。有关更多信息，请参阅 CloudWatchFullAccessV2。	2023 年 8 月 1 日
AWSServiceRoleForInternetMonitor — 更新到现有政策	Amazon CloudWatch 网络监测仪添加了监控网络负载均衡器资源的新权限。 需要 `elasticloadbalancing:DescribeLoadBalancers` 和 `ec2:DescribeNetworkInterfaces` 权限，以便网络监测仪可以通过分析 NLB 资源的流日志来监测客户的网络负载均衡器流量。 有关更多信息，请参阅 使用 Amazon CloudWatch 网络监测仪。	2023 年 7 月 15 日

更改	描述	日期
CloudWatchReadOnlyAccess – 对现有策略的更新	CloudWatch 添加了对 CloudWatchReadOnlyAccess 的权限。 增加了 `logs:StartLiveTail` 和 `logs:StopLiveTail` 权限，以便使用此策略的用户可以使用控制台启动和停止 CloudWatch Logs Live Tail 会话。有关更多信息，请参阅 使用 Live Tail 近乎实时地查看日志。	2023 年 6 月 6 日
CloudWatchCrossAccountSharingConfiguration – 新策略	CloudWatch 添加了新策略，助力您管理用于分享 CloudWatch 指标的 CloudWatch 跨账户可观测性链接。 有关更多信息，请参阅 CloudWatch 跨账户可观测性。	2022 年 11 月 27 日
OAMFullAccess – 新策略	CloudWatch 添加了新策略，助力您全面管理 CloudWatch 跨账户可观测性链接和汇点。 有关更多信息，请参阅 CloudWatch 跨账户可观测性。	2022 年 11 月 27 日

更改	描述	日期
OAMReadOnlyAccess – 新策略	CloudWatch 添加了新策略，助力您查看关于 CloudWatch 跨账户可观测性链接和汇点的信息。 有关更多信息，请参阅 CloudWatch 跨账户可观测性。	2022 年 11 月 27 日
CloudWatchFullAccess – 对现有策略的更新	CloudWatch 添加了对 CloudWatchFullAccess 的权限。 添加了 `oam:ListSinks` 和 `oam:ListAttachedLinks` 权限，以便使用此策略的用户可以借助控制台在 CloudWatch 跨账户可观察性中查看源账户共享的数据。	2022 年 11 月 27 日
CloudWatchReadOnlyAccess – 对现有策略的更新	CloudWatch 添加了对 CloudWatchReadOnlyAccess 的权限。 添加了 `oam:ListSinks` 和 `oam:ListAttachedLinks` 权限，以便使用此策略的用户可以借助控制台在 CloudWatch 跨账户可观察性中查看源账户共享的数据。	2022 年 11 月 27 日

更改	描述	日期
AmazonCloudWatchRUMServiceRolePolicy – 对现有策略的更新	CloudWatch RUM 更新了 AmazonCloudWatchRUMServiceRolePolicy 中的一个条件键。 `"Condition": { "StringEquals": { "cloudwatch:namespace": "AWS/RUM" } }` 条件键已进行如下更改，以便 CloudWatch RUM 可以将自定义指标发送到自定义指标命名空间。 `"Condition": { "StringLike": { "cloudwatch:namespace": [ "RUM/CustomMetrics/*", "AWS/RUM" ] } }`	2023 年 2 月 2 日

更改	描述	日期
AmazonCloudWatchRUMReadOnlyAccess – 更新后的策略	CloudWatch 向 AmazonCloudWatchRUMReadOnlyAccess 策略添加了权限。 添加了 `rum:ListRumMetricsDestinations` 和 `rum:BatchGetRumMetricsDefinitions` 权限，这样 CloudWatch RUM 就可以向 CloudWatch 和 Evidently 发送扩展指标。	2022 年 10 月 27 日
AmazonCloudWatchRUMServiceRolePolicy – 对现有策略的更新	CloudWatch RUM 向 AmazonCloudWatchRUMServiceRolePolicy 添加了权限。 添加了 `cloudwatch:PutMetricData` 权限，这样 CloudWatch RUM 就可以向 CloudWatch 发送扩展指标。	2022 年 10 月 26 日
CloudWatchEvidentlyReadOnlyAccess – 对现有策略的更新	CloudWatch Evidently 添加了对 CloudWatchEvidentlyReadOnlyAccess 的权限。 添加了 `evidently:GetSegment` 、`evidently:ListSegments` 和 `evidently:ListSegmentReferences` 权限，以便使用此策略的用户可以看到已创建的 Evidently 受众细分。	2022 年 8 月 12 日

更改	描述	日期
CloudWatchSyntheticsFullAccess – 对现有策略的更新	CloudWatch Synthetics 添加了对 CloudWatchSyntheticsFullAccess 的权限。 添加了 `lambda:DeleteFunction` 和 `lambda:DeleteLayerVersion` 权限，以便 CloudWatch Synthetics 可在 Canary 时删除相关资源。添加了 `iam:ListAttachedRolePolicies` ，以便客户可以查看附加到 Canary IAM 角色的策略。	2022 年 5 月 6 日
AmazonCloudWatchRUMFullAccess – 新策略	CloudWatch 添加了一项启用对 CloudWatch RUM 的全面管理的新策略。 CloudWatch RUM 允许您对 Web 应用程序执行真实的用户监控。有关更多信息，请参阅 CloudWatch RUM。	2021 年 11 月 29 日
AmazonCloudWatchRUMReadOnlyAccess – 新策略	CloudWatch 添加了一项启用对 CloudWatch RUM 的只读访问的新策略。 CloudWatch RUM 允许您对 Web 应用程序执行真实的用户监控。有关更多信息，请参阅 CloudWatch RUM。	2021 年 11 月 29 日

更改	描述	日期
CloudWatchEvidentlyFullAccess – 新策略	CloudWatch 添加了一项启用对 CloudWatch Evidently 的全面管理的新策略。 CloudWatch Evidently 允许您对 Web 应用程序执行 A/B 实验，并逐步执行这些实验。有关更多信息，请参阅 使用 CloudWatch Evidently 执行启动和 A/B 实验。	2021 年 11 月 29 日
CloudWatchEvidentlyReadOnlyAccess – 新策略	CloudWatch 添加了一项启用对 CloudWatch Evidently 的只读访问的新策略。 CloudWatch Evidently 允许您对 Web 应用程序执行 A/B 实验，并逐步执行这些实验。有关更多信息，请参阅 使用 CloudWatch Evidently 执行启动和 A/B 实验。	2021 年 11 月 29 日
AWSServiceRoleForCloudWatchRUM – 新的托管式策略	CloudWatch 添加了一项新的服务相关角色的策略，以允许 CloudWatch RUM 将监控数据发布给其他相关 AWS 服务。	2021 年 11 月 29 日

更改	描述	日期
CloudWatchSyntheticsFullAccess – 对现有策略的更新	CloudWatch Synthetics 向 CloudWatchSyntheticsFullAccess 添加了权限，还更改了一个权限的范围。 添加了 `kms:ListAliases` 权限，以便用户可以列出可用于加密 canary 构件的可用 AWS KMS 密钥。添加了 `kms:DescribeKey` 权限，以便用户可以查看将用于加密 canary 构件的密钥的详细信息。此外，还添加了 `kms:Decrypt` 权限，以便用户能够解密 canary 构件。此解密功能仅限用于 Amazon S3 存储桶中的资源。 `s3:GetBucketLocation` 权限的 `Resource` 范围从 `*` 更改为了 `arn:aws:s3:::*` 。	2021 年 9 月 29 日
CloudWatchSyntheticsFullAccess – 对现有策略的更新	CloudWatch Synthetics 添加了一个对 CloudWatchSyntheticsFullAccess 策略的权限。 `lambda:UpdateFunctionCode` 权限，以便使用此策略的用户可以更改 Canary 的运行时版本。	2021 年 7 月 20 日
AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy – 新托管式策略	CloudWatch 添加了一个新的托管式 IAM 策略，以允许 CloudWatch 在 AWS Systems Manager Incident Manager 中创建事件。	2021 年 5 月 10 日

更改	描述	日期
CloudWatchAutomaticDashboardsAccess – 对现有策略的更新	CloudWatch 添加了一个对 CloudWatchAutomaticDashboardsAccess 托管式策略的权限。`synthetics:DescribeCanariesLastRun` 权限添加到此策略中，以使跨账户控制面板用户能够查看有关 CloudWatch Synthetics canary 运行的详细信息。	2021 年 4 月 20 日
CloudWatch 开始跟踪更改	CloudWatch 开始跟踪其 AWS 托管式策略的更改。	2021 年 4 月 14 日

使用条件键限制对 CloudWatch 命名空间的访问

使用 IAM 条件键限制用户仅在指定的 CloudWatch 命名空间中发布指标。

仅允许在一个命名空间中发布

以下策略将用户限制为仅在名为 `MyCustomNamespace` 的命名空间中发布指标。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "MyCustomNamespace"
            }
        }
    }
}
```

排除从命名空间发布

以下策略允许用户在除 CustomNamespace2 之外的任何命名空间中发布指标。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        },
        {
            "Effect": "Deny",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CustomNamespace2"
                }
            }
        }
    ]
}
```

使用条件键限制 Contributor Insights 用户对日志组的访问

要在 Contributor Insights 中创建规则并查看其结果，用户必须具有 cloudwatch:PutInsightRule 权限。默认情况下，具有此权限的用户可以创建 Contributor Insights 规则，用于评估 CloudWatch Logs 中的任何日志组，然后查看结果。结果可以包含这些日志组的贡献者数据。

您可以使用条件密钥创建 IAM 策略，以授予用户为某些日志组编写 Contributor Insights 规则的权限，同时阻止用户为其他日志组编写规则和查看此数据。

有关 IAM 策略中 Condition 元素的更多信息，请参阅 IAM JSON 策略元素：条件。

仅允许对特定日志组的编写规则和查看结果的访问权限

以下策略允许用户访问对名为 AllowedLogGroup 的日志组和名称以 AllowedWildCard 开头的所有日志组的编写规则和查看结果的访问权限。它不授予对其他日志组的编写规则或查看规则结果的访问权限。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "AllowCertainLogGroups",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "AllowedLogGroup",
                        "AllowedWildcard*"
                    ]
                }
            }
        }
    ]
}
```

拒绝为特定日志组编写规则，但允许为所有其他日志组编写规则

以下策略显式拒绝用户对名为 ExplicitlyDeniedLogGroup 的日志组编写规则和查看结果的访问权限，但允许对所有其他日志组编写规则和查看规则结果。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInsightRulesOnLogGroupsByDefault",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*"

        },
        {
            "Sid": "ExplicitDenySomeLogGroups",
            "Effect": "Deny",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "/test/alpine/ExplicitlyDeniedLogGroup"
                    ]
```

```
                }
            }
        }
    ]
}
```

使用条件键限制告警操作

当 CloudWatch 告警更改状态时，它们可以执行不同的操作，例如停止和终止 EC2 实例以及执行 Systems Manager 操作。当告警变为任何状态（包括 ALARM（告警）、OK（正常）或 INSUFFICIENT_DATA（数据不足））时，可以启动这些操作。

使用 `cloudwatch:AlarmActions` 条件键，以允许用户创建告警，这些告警只能在告警状态发生变化时执行您指定的操作。例如，您可以允许用户创建只能执行非 EC2 操作的告警。

允许用户创建只能发送 Amazon SNS 通知或执行 Systems Manager 操作的告警

以下策略限制用户只能创建发送 Amazon SNS 通知或执行 Systems Manager 操作的告警。用户不能创建执行 EC2 操作的告警。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:AlarmActions": [
                        "arn:aws:sns:*",
                        "arn:aws:ssm:*"
                    ]
                }
            }
        }
    ]
}
```

为 CloudWatch 使用服务相关角色

Amazon CloudWatch 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 CloudWatch 直接相关。服务相关角色由 CloudWatch 预定义，并包含该服务代表您调用其他 AWS 服务所需的一切权限。

CloudWatch 中的一个服务相关角色将设置可终止、停止或重启 Amazon EC2 实例的 CloudWatch 告警，而无需您手动添加必要的权限。另一个服务相关角色可让监控账户访问您指定的其他账户的 CloudWatch 数据，从而构建跨账户跨区域的控制面板。

CloudWatch 定义这些服务相关角色的权限，除非另行定义，否则仅 CloudWatch 能够代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有在首先删除角色的相关资源后，才能删除角色。此限制将保护您的 CloudWatch 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的 AWS 服务并查找 Service-Linked Role（服务相关角色）列中显示为 Yes（是）的服务。选择是和链接，查看该服务的服务相关角色文档。

CloudWatch EC2 操作的服务相关角色权限

CloudWatch 使用名为 AWSServiceRoleForCloudWatchEvents 的服务相关角色 – CloudWatch 使用此服务相关角色执行 Amazon EC2 告警操作。

AWSServiceRoleForCloudWatchEvents 服务相关角色信任 CloudWatch Events 服务来代入该角色。CloudWatch Events 在被告警调起时，调用终止、停止或重启实例操作。

AWSServiceRoleForCloudWatchEvents 服务相关角色权限策略允许 CloudWatch Events 对 Amazon EC2 实例完成以下操作：

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`
- `ec2:DescribeInstanceRecoveryAttribute`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceStatus`

AWSServiceRoleForCloudWatchCrossAccount 服务相关角色权限策略允许 CloudWatch 完成以下操作：

- sts:AssumeRole

CloudWatch Application Signals 的服务相关角色权限

CloudWatch Application Signals 使用名为 AWSServiceRoleForCloudWatchApplicationSignals 的服务相关角色。CloudWatch 使用这个服务相关角色从您为 CloudWatch Application Signals 启用的应用程序中收集 CloudWatch Logs 数据、X-Ray 跟踪数据、CloudWatch 指标数据和标记数据。

AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色信任 CloudWatch Application Signals 来代入该角色。Application Signals 会从您的账户收集日志、跟踪、指标和标签数据。

AWSServiceRoleForCloudWatchApplicationSignals 附加有 IAM 策略，此政策名为 CloudWatchApplicationSignalsServiceRolePolicy。此策略授予 CloudWatch Application Signals 从其他相关 AWS 服务收集监控和标记数据的权限。其中包含允许 Application Signals 完成以下操作的权限：

- xray:GetServiceGraph
- logs:StartQuery
- logs:GetQueryResults
- cloudwatch:GetMetricData
- cloudwatch:ListMetrics
- tag:GetResources

CloudWatchSyntheticsReadOnlyAccess 的完整内容如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "XRayPermission",
            "Effect": "Allow",
            "Action": [
                "xray:GetServiceGraph"
            ],
            "Resource": [
                "*"
```

```
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWLogsPermission",
            "Effect": "Allow",
            "Action": [
                "logs:StartQuery",
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/appsignals/*:*",
                "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWListMetricsPermission",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListMetrics"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "CWGetMetricDataPermission",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "TagsPermission",
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        }
    ]
}
```

CloudWatch 告警 Systems Manager OpsCenter 操作的服务相关角色权限

CloudWatch 使用名为 AWSServiceRoleForCloudWatchAlarms_ActionSSM 的服务相关角色 – CloudWatch 在告警进入 ALARM（告警）状态时使用此服务相关角色执行 Systems Manager OpsCenter 操作。

AWSServiceRoleForCloudWatchAlarms_ActionSSM 服务相关角色信任 CloudWatch 服务来代入该角色。CloudWatch 告警在被告警调用时，将调用 Systems Manager OpsCenter 操作。

AWSServiceRoleForCloudWatchAlarms_ActionSSM 服务相关角色权限策略允许 Systems Manager 完成以下操作：

- `ssm:CreateOpsItem`

CloudWatch 告警 Systems Manager Incident Manager 操作的服务相关角色权限

CloudWatch 使用名为 AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents 的服务相关角色 – CloudWatch 在告警进入 ALARM（告警）状态时使用此服务相关角色启动 Incident Manager 事件。

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents 服务相关角色信任 CloudWatch 服务来代入该角色。CloudWatch 告警在被告警调用时，将调用 Systems Manager Incident Manager 操作。

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents 服务相关角色权限策略允许 Systems Manager 完成以下操作：

- `ssm-incidents:StartIncident`

CloudWatch 跨账户跨区域的服务相关角色权限

CloudWatch 使用名为 AWSServiceRoleForCloudWatchCrossAccount 的服务相关角色 – CloudWatch 使用此角色访问您指定的其他 AWS 账户中的 CloudWatch 数据。SLR 仅提供代入角色权限以允许 CloudWatch 服务代入共享账户中的角色。它是提供对数据的访问权限的共享角色。

AWSServiceRoleForCloudWatchCrossAccount 服务相关角色权限策略允许 CloudWatch 完成以下操作：

- `sts:AssumeRole`

AWSServiceRoleForCloudWatchCrossAccount 服务相关角色信任 CloudWatch 服务来代入该角色。

CloudWatch 数据库性能详情的服务相关角色权限

CloudWatch 使用名为 AWSServiceRoleForCloudWatchMetrics_DbPerfInsights 的服务相关角色。– CloudWatch 使用此角色检索用于创建警报和快照的性能详情指标。

AWSServiceRoleForCloudWatchMetrics_DbPerfInsights 服务相关角色附加了 `AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy` IAM 策略。该策略的内容如下所示：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "pi:GetResourceMetrics"
   ],
   "Resource": "*",
   "Condition": {
```

```
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

AWSServiceRoleForCloudWatchMetrics_DbPerfInsights 服务相关角色信任 CloudWatch 服务来代入该角色。

为 CloudWatch 创建服务相关角色

您无需手动创建这两个服务相关角色。当您首次在 AWS Management Console、IAM CLI 或 IAM API 中创建告警时，CloudWatch 会为您创建 AWSServiceRoleForCloudWatchEvents 和 AWSServiceRoleForCloudWatchAlarms_ActionSSM。

首次启用服务和拓扑发现时，Application Signals 会为您创建 AWSServiceRoleForCloudWatchApplicationSignals。

当您首次启用一个账户作为跨账户跨区域功能的监控账户时，CloudWatch 会为您创建 AWSServiceRoleForCloudWatchCrossAccount。

当您第一次创建使用 DB_PERF_INSIGHTS 指标数学函数的警报时，CloudWatch 会为您创建 AWSServiceRoleForCloudWatchMetrics_DbPerfInsights。

有关更多信息，请参阅《IAM 用户指南》中的 创建服务相关角色。

为 CloudWatch 编辑服务相关角色

CloudWatch 不允许您编辑 AWSServiceRoleForCloudWatchEvents、AWSServiceRoleForCloudWatchAlarms_ActionSSM、AWSServic 或 AWSServiceRoleForCloudWatchMetrics_DbPerfInsights 角色。在创建这些角色后，您无法更改这些角色的名称，因为可能有不同的实体引用它们。但是，您可以使用 IAM 编辑这些角色的描述。

编辑服务相关角色描述（IAM 控制台）

您可以使用 IAM 控制台编辑服务相关角色的描述。

编辑服务相关角色的描述（控制台）

1. 在 IAM 控制台的导航窗格中，选择角色。

2. 以下代码示例显示如何将 IAM 策略附加到用户。
3. 在 Role description 的最右侧，选择 Edit。
4. 在框中键入新描述，然后选择 Save (保存)。

编辑服务相关角色描述 (AWS CLI)

您可以从 AWS Command Line Interface 使用 IAM 命令编辑服务相关角色的描述。

更改服务相关角色描述 (AWS CLI)

1. (可选) 要查看角色的当前描述，请使用以下命令：

```
$ aws iam get-role --role-name role-name
```

可以在 AWS CLI 命令中使用角色名称（而不是 ARN）引用角色。例如，如果某个角色的 ARN 为 `arn:aws:iam::123456789012:role/myrole`，则将该角色称为 **myrole**。

2. 要更新服务相关角色的描述，请使用以下命令：

```
$ aws iam update-role-description --role-name role-name --description description
```

编辑服务相关角色描述 (IAM API)

您可以使用 IAM API 编辑服务相关角色的描述。

更改服务相关角色的描述（API）

1. (可选) 要查看角色的当前描述，请使用以下命令：

GetRole

2. 要更新角色的描述，请使用以下命令：

UpdateRoleDescription

为 CloudWatch 删除服务相关角色

如果您不再具有自动停止、终止或重启 EC2 实例的警报，我们建议您删除 AWSServiceRoleForCloudWatchEvents 角色。

如果您不再具有执行 Systems Manager OpsCenter 操作的告警，我们建议您删除 AWSServiceRoleForCloudWatchAlarms_ActionSSM 角色。

如果您删除使用 DB_PERF_INSIGHTS 指标数学函数的所有警报，则建议您删除 AWSServiceRoleForCloudWatchMetrics_DbPerfInsights 服务相关角色。

这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能将其删除。

清除服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

在 IAM 控制台中检查服务相关角色是否具有活动会话

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。选择 AWSServiceRoleForCloudWatchEvents 角色的名称（不是复选框）。
3. 在选定角色的摘要页上，选择访问顾问并查看服务相关角色的近期活动。

> **Note**
>
> 如果您不确定 CloudWatch 是否正在使用 AWSServiceRoleForCloudWatchEvents 角色，请尝试删除该角色。如果服务正在使用该角色，则删除操作会失败，并且您可以查看正在使用该角色的 区域。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。

删除服务相关角色（IAM 控制台）

您可以使用 IAM 控制台删除服务相关角色。

删除服务相关角色（控制台）

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。选中要删除的角色名称旁的复选框，而不是名称或行本身。
3. 对于角色操作，请选择删除角色。

4. 在确认对话框中，查看上次访问服务数据，该数据显示每个选定角色上次访问AWS服务的时间。这样可帮助您确认角色当前是否处于活动状态。要继续，请选择 Yes, Delete（是的，删除）。
5. 监视 IAM 控制台通知，以监控服务相关角色的删除进度。由于 IAM 服务相关角色的删除是异步操作，因此，在提交要删除的角色后，删除任务可能会成功，也可能会失败。如果该任务失败，请从通知中选择查看详细信息或查看资源以了解删除失败的原因。如果因角色正在使用服务中的资源而导致删除操作失败，则失败原因将包含一个资源列表。

删除服务相关角色 (AWS CLI)

您可以从 AWS Command Line Interface 使用 IAM 命令删除服务相关角色。

删除服务相关角色 (AWS CLI)

1. 如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 `deletion-task-id` 以检查删除任务的状态。键入以下命令以提交服务相关角色的删除请求：

   ```
   $ aws iam delete-service-linked-role --role-name service-linked-role-name
   ```

2. 键入以下命令以检查删除任务的状态：

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   删除任务的状态可能是 `NOT_STARTED`、`IN_PROGRESS`、`SUCCEEDED` 或 `FAILED`。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

删除服务相关角色（IAM API）

您可以使用 IAM API 删除服务相关角色。

删除服务相关角色（API）

1. 要提交服务相关角色的删除请求，请调用 DeleteServiceLinkedRole。在请求中，指定您要删除的角色名称。

   如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 `DeletionTaskId` 以检查删除任务的状态。

2. 要检查删除的状态，请调用 GetServiceLinkedRoleDeletionStatus。在请求中，指定 DeletionTaskId。

   删除任务的状态可能是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

对 AWS 服务相关角色的 CloudWatch 更新

查看有关 CloudWatch 的 AWS 托管式策略更新的详细信息（从该服务开始跟踪这些更改开始）。有关此页面更改的自动提示，请订阅 CloudWatch 文档历史记录页面上的 RSS 源。

更改	描述	日期
AWSServiceRoleForCloudWatchApplicationSignals – 更新为服务相关角色策略的权限	CloudWatch 将更多日志组添加到此角色授予的 logs:StartQuery 和 logs:GetQueryResults 权限的范围。	2024 年 4 月 24 日
AWSServiceRoleForCloudWatchApplicationSignals：新的服务相关角色	CloudWatch 添加了这个新的服务相关角色，以允许 CloudWatch Application Signals 收集 CloudWatch Logs 数据、X-Ray 跟踪数据、CloudWatch 指标数据，以及您为 CloudWatch Application Signals 启用的应用程序中的标记数据。	2023 年 11 月 9 日
AWSServiceRoleForCloudWatchMetrics_DbPerfInsights – 新的服务相关角色	CloudWatch 添加了这个新的服务相关角色，允许 CloudWatch 获取用于警报和快照的性能详情指标。此角色附加了一个 IAM 策略，该策略	2023 年 9 月 13 日

更改	描述	日期
	授予 CloudWatch 代表您获取性能详情指标的权限。	
AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – 新的服务相关角色	CloudWatch 添加了新的服务相关角色，以允许 CloudWatch 在 AWS Systems Manager Incident Manager 中创建事件。	2021 年 4 月 26 日
CloudWatch 开始跟踪更改	CloudWatch 开始跟踪其服务相关角色的更改。	2021 年 4 月 26 日

对 CloudWatch RUM 使用服务相关角色

CloudWatch RUM 使用了 AWS Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 RUM 直接相关。服务相关角色是由 RUM 预定义的，包含该服务代表您调用其他 AWS 服务所需的所有权限。

RUM 定义服务相关角色的权限，除非另有定义，否则只有 RUM 可以承担该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有先删除角色的相关资源，才能删除角色。此限制可保护您的 RUM 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务，并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

RUM 的服务相关角色权限

RUM 对您为其启用了 X-Ray 跟踪的应用程序监控使用名为 AWSServiceRoleForCloudWatchRUM 的服务相关角色 – 此角色允许 RUM 将 AWS X-Ray 跟踪数据发送到您的账户。

AWSServiceRoleForCloudWatchRUM 服务相关角色信任 X-Ray 服务来代入该角色。X-Ray 将跟踪数据发送到您的账户。

AWSServiceRoleForCloudWatchRUM 服务相关角色附加有名为 AmazonCloudWatchRUMServiceRolePolicy 的 IAM 策略。此策略向 CloudWatch RUM 授予将监控数据发布到其他相关 AWS 服务的权限。其中包括允许 RUM 完成以下操作的权限：

- `xray:PutTraceSegments`
- `cloudwatch:PutMetricData`

AmazonCloudWatchRUMServiceRolePolicy 的完整内容如下所示。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

为 RUM 创建服务相关角色

您无需手动为 CloudWatch RUM 创建服务相关角色。当您首次创建启用了 X-Ray 跟踪的应用程序监控，或更新应用程序监控以使用 X-Ray 跟踪时，RUM 会为您创建 AWSServiceRoleForCloudWatchRUM。

有关更多信息，请参阅《IAM 用户指南》中的创建服务相关角色。

编辑 RUM 的服务相关角色

CloudWatch RUM 不允许您编辑 AWSServiceRoleForCloudWatchRUM 角色。在创建这些角色后，您无法更改这些角色的名称，因为可能有不同的实体引用它们。但是，您可以使用 IAM 编辑这些角色的描述。

编辑服务相关角色描述（IAM 控制台）

您可以使用 IAM 控制台编辑服务相关角色的描述。

编辑服务相关角色的描述（控制台）

1. 在 IAM 控制台的导航窗格中，选择角色。
2. 以下代码示例显示如何将 IAM 策略附加到用户。
3. 在 Role description 的最右侧，选择 Edit。
4. 在框中键入新描述，然后选择 Save (保存)。

编辑服务相关角色描述 (AWS CLI)

您可以从 AWS Command Line Interface 使用 IAM 命令编辑服务相关角色的描述。

更改服务相关角色描述 (AWS CLI)

1. (可选) 要查看角色的当前描述，请使用以下命令：

   ```
   $ aws iam get-role --role-name role-name
   ```

   可以在 AWS CLI 命令中使用角色名称（而不是 ARN）引用角色。例如，如果某个角色的 ARN 为 `arn:aws:iam::123456789012:role/myrole`，则将该角色称为 **myrole**。
2. 要更新服务相关角色的描述，请使用以下命令：

```
$ aws iam update-role-description --role-name role-name --description description
```

编辑服务相关角色描述 (IAM API)

您可以使用 IAM API 编辑服务相关角色的描述。

更改服务相关角色的描述（API）

1. (可选) 要查看角色的当前描述，请使用以下命令：

   GetRole
2. 要更新角色的描述，请使用以下命令：

   UpdateRoleDescription

删除 RUM 的服务相关角色

如果您不再拥有启用了 X-Ray 的应用程序监控，我们建议您删除 AWSServiceRoleForCloudWatchRUM 角色。

这样就没有未被主动监控或维护的未使用实体。但是，您必须先清除您的服务相关角色，然后才能将其删除。

清除服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

在 IAM 控制台中检查服务相关角色是否具有活动会话

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。选择 AWSServiceRoleForCloudWatchRUM 角色的名称（而不是复选框）。
3. 在选定角色的摘要页上，选择访问顾问并查看服务相关角色的近期活动。

> **Note**
>
> 如果您不确定 RUM 是否正在使用 AWSServiceRoleForCloudWatchRUM 角色，请尝试删除该角色。如果服务正在使用该角色，则删除操作会失败，并且您可以查看正在使用该角色的 区域。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。

删除服务相关角色（IAM 控制台）

您可以使用 IAM 控制台删除服务相关角色。

删除服务相关角色（控制台）

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。选中要删除的角色名称旁的复选框，而不是名称或行本身。
3. 对于角色操作，请选择删除角色。
4. 在确认对话框中，查看上次访问服务数据，该数据显示每个选定角色上次访问AWS服务的时间。这样可帮助您确认角色当前是否处于活动状态。要继续，请选择 Yes, Delete（是的，删除）。
5. 监视 IAM 控制台通知，以监控服务相关角色的删除进度。由于 IAM 服务相关角色的删除是异步操作，因此，在提交要删除的角色后，删除任务可能会成功，也可能会失败。如果该任务失败，请从通知中选择查看详细信息或查看资源以了解删除失败的原因。如果因角色正在使用服务中的资源而导致删除操作失败，则失败原因将包含一个资源列表。

删除服务相关角色 (AWS CLI)

您可以从 AWS Command Line Interface 使用 IAM 命令删除服务相关角色。

删除服务相关角色 (AWS CLI)

1. 如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 `deletion-task-id` 以检查删除任务的状态。键入以下命令以提交服务相关角色的删除请求：

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. 键入以下命令以检查删除任务的状态：

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

删除任务的状态可能是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

删除服务相关角色（IAM API）

您可以使用 IAM API 删除服务相关角色。

删除服务相关角色（API）

1. 要提交服务相关角色的删除请求，请调用 DeleteServiceLinkedRole。在请求中，指定您要删除的角色名称。

   如果服务相关角色正被使用或具有关联的资源，则无法删除它，因此您必须提交删除请求。如果不满足这些条件，该请求可能会被拒绝。您必须从响应中捕获 DeletionTaskId 以检查删除任务的状态。

2. 要检查删除的状态，请调用 GetServiceLinkedRoleDeletionStatus。在请求中，指定 DeletionTaskId。

   删除任务的状态可能是 NOT_STARTED、IN_PROGRESS、SUCCEEDED 或 FAILED。如果删除失败，则调用会返回失败的原因，以便您进行问题排查。

在 CloudWatch Application Insights 中使用服务相关角色

CloudWatch Application Insights 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 CloudWatch Application Insights 直接相关。服务相关角色由 CloudWatch Application Insights 预定义，并包含该服务代表您调用其他 AWS 服务所需的一切权限。

服务相关角色更方便设置 CloudWatch Application Insights，因为无需手动添加必要权限。CloudWatch Application Insights 定义其服务相关角色的权限，除非另外定义，否则只有 CloudWatch Application Insights 可以代入其角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

有关支持服务相关角色的其他服务的信息，请参阅使用 IAM 的 AWS 服务并查找 Service-Linked Role（服务相关角色）列中显示为 Yes（是）的服务。选择是链接，查看该服务的服务相关角色文档。

CloudWatch Application Insights 的服务相关角色权限

CloudWatch Application Insights 使用名为 AWSServiceRoleForApplicationInsights 的服务相关角色。Application Insights 使用此角色执行操作，例如分析客户的资源组、创建 CloudFormation 堆栈以创建有关指标的告警，以及在 EC2 实例上配置 CloudWatch 代理。服务相关角色附加了 IAM policy，名为 `CloudwatchApplicationInsightsServiceLinkedRolePolicy`。有关此策略的更新，请参阅 对 AWS 托管式策略的 Application Insights 更新。

角色权限策略允许 CloudWatch Application Insights 对资源完成以下操作。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatch",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:PutAnomalyDetector",
        "cloudwatch:DeleteAnomalyDetector",
        "cloudwatch:DescribeAnomalyDetectors"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridge",
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudFormation",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:CreateStack",
        "cloudFormation:UpdateStack",
        "cloudFormation:DeleteStack",
        "cloudFormation:DescribeStackResources",
        "cloudFormation:UpdateTerminationProtection"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "CloudFormationStacks",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:DescribeStacks",
        "cloudFormation:ListStackResources",
        "cloudFormation:ListStacks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Tag",
      "Effect": "Allow",
```

```
      "Action": [
        "tag:GetResources"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ResourceGroups",
      "Effect": "Allow",
      "Action": [
        "resource-groups:ListGroupResources",
        "resource-groups:GetGroupQuery",
        "resource-groups:GetGroup"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationInsightsResourceGroup",
      "Effect": "Allow",
      "Action": [
        "resource-groups:CreateGroup",
        "resource-groups:DeleteGroup"
      ],
      "Resource": [
        "arn:aws:resource-groups:*:*:group/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "ElasticLoadBalancing",
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AutoScaling",
```

```
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMParameter",
      "Effect": "Allow",
      "Action": [
        "ssm:PutParameter",
        "ssm:DeleteParameter",
        "ssm:AddTagsToResource",
        "ssm:RemoveTagsFromResource",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-ApplicationInsights-*"
    },
    {
      "Sid": "SSMAssociation",
      "Effect": "Allow",
      "Action": [
        "ssm:CreateAssociation",
        "ssm:UpdateAssociation",
        "ssm:DeleteAssociation",
        "ssm:DescribeAssociation"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:association/*",
        "arn:aws:ssm:*:*:managed-instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Sid": "SSMOpsItem",
      "Effect": "Allow",
      "Action": [
        "ssm:GetOpsItem",
```

```
        "ssm:CreateOpsItem",
        "ssm:DescribeOpsItems",
        "ssm:UpdateOpsItem",
        "ssm:DescribeInstanceInformation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMTags",
      "Effect": "Allow",
      "Action": [
        "ssm:AddTagsToResource"
      ],
      "Resource": "arn:aws:ssm:*:*:opsitem/*"
    },
    {
      "Sid": "SSMGetCommandInvocation",
      "Effect": "Allow",
      "Action": [
        "ssm:ListCommandInvocations",
        "ssm:GetCommandInvocation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMSendCommand",
      "Effect": "Allow",
      "Action": "ssm:SendCommand",
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-CheckPerformanceCounterSets",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AWSEC2-DetectWorkload",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Sid": "EC2",
      "Effect": "Allow",
      "Action": [
```

```
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeNatGateways"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "RDS",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Lambda",
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions",
        "lambda:GetFunctionConfiguration",
        "lambda:ListEventSourceMappings"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridgeManagedRule",
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:DeleteRule"
      ],
      "Resource": [
```

```
        "arn:aws:events:*:*:rule/AmazonCloudWatch-ApplicationInsights-*"
      ]
    },
    {
      "Sid": "XRay",
      "Effect": "Allow",
      "Action": [
        "xray:GetServiceGraph",
        "xray:GetTraceSummaries",
        "xray:GetTimeSeriesServiceStatistics",
        "xray:GetTraceGraph"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "DynamoDB",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables",
        "dynamodb:DescribeTable",
        "dynamodb:DescribeContributorInsights",
        "dynamodb:DescribeTimeToLive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationAutoscaling",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalableTargets"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3",
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
```

```
        "s3:GetMetricsConfiguration",
        "s3:GetReplicationConfiguration"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "States",
      "Effect": "Allow",
      "Action": [
        "states:ListStateMachines",
        "states:DescribeExecution",
        "states:DescribeStateMachine",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "APIGateway",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECS",
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:DescribeTaskSets",
        "ecs:ListClusters",
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "ecs:ListTasks"
```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECSCluster",
      "Effect": "Allow",
      "Action": [
        "ecs:UpdateClusterSettings"
      ],
      "Resource": [
        "arn:aws:ecs:*:*:cluster/*"
      ]
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:DescribeFargateProfile",
        "eks:DescribeNodegroup",
        "eks:ListClusters",
        "eks:ListFargateProfiles",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SNS",
      "Effect": "Allow",
      "Action": [
        "sns:GetSubscriptionAttributes",
        "sns:GetTopicAttributes",
        "sns:GetSMSAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics"
      ],
      "Resource": [
        "*"
```

```
      ]
    },
    {
      "Sid": "SQS",
      "Effect": "Allow",
      "Action": [
        "sqs:ListQueues"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsDeleteSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:DeleteSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*"
      ]
    },
    {
      "Sid": "CloudWatchLogsCreateSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:PutSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:*:destination:AmazonCloudWatch-ApplicationInsights-
LogIngestionDestination*"
      ]
    },
    {
      "Sid": "EFS",
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Route53",
```

```
      "Effect": "Allow",
      "Action": [
        "route53:GetHostedZone",
        "route53:GetHealthCheck",
        "route53:ListHostedZones",
        "route53:ListHealthChecks",
        "route53:ListQueryLoggingConfigs"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Route53Resolver",
      "Effect": "Allow",
      "Action": [
        "route53resolver:ListFirewallRuleGroupAssociations",
        "route53resolver:GetFirewallRuleGroup",
        "route53resolver:ListFirewallRuleGroups",
        "route53resolver:ListResolverEndpoints",
        "route53resolver:GetResolverQueryLogConfig",
        "route53resolver:ListResolverQueryLogConfigs",
        "route53resolver:ListResolverQueryLogConfigAssociations",
        "route53resolver:GetResolverEndpoint",
        "route53resolver:GetFirewallRuleGroupAssociation"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 CloudWatch Application Insights 创建服务相关角色

您无需手动创建服务相关角色。当您在 AWS Management Console 中创建新的 Application Insights 应用程序时，CloudWatch Application Insights 将为您创建服务相关角色。

如果删除该服务相关角色，然后希望再次创建该角色，您可以使用相同的过程在您的账户中重新创建该角色。当您创建新的 Application Insights 应用程序时，CloudWatch Application Insights 将为您再次创建服务相关角色。

为 CloudWatch Application Insights 编辑服务相关角色

CloudWatch Application Insights 不允许您编辑 AWSServiceRoleForApplicationInsights 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

为 CloudWatch Application Insights 删除服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样，您就可以避免使用当前未监控或维护的未使用实体。不过，您必须先删除 Application Insights 中的所有应用程序，然后才能手动删除该角色。

> **Note**
>
> 在尝试删除资源时，如果 CloudWatch Application Insights 服务正在使用该角色，删除可能会失败。如果发生这种情况，请等待几分钟后重试。

若要删除 AWSServiceRoleForApplicationInsights 使用的 CloudWatch Application Insights 资源

- 删除所有 CloudWatch Application Insights 应用程序。有关更多信息，请参阅《CloudWatch Application Insights 用户指南》中的“删除应用程序”。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForApplicationInsights 服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

CloudWatch Application Insights 服务相关角色支持的区域

CloudWatch Application Insights 支持在服务可用的所有 AWS 区域中使用服务相关角色。有关更多信息，请参阅 CloudWatch Application Insights 区域和端点。

适用于 Amazon CloudWatch Application Insights 的 AWS 托管式策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户托管式策略来进一步减少权限。

您无法更改 AWS 托管式策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务 启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管式策略：CloudWatchApplicationInsightsFullAccess

您可以将 `CloudWatchApplicationInsightsFullAccess` 策略附加到 IAM 身份。

此策略授予管理权限，允许完全访问 Application Insights 功能。

权限详细信息

该策略包含以下权限。

- `applicationinsights` – 允许完全访问 Application Insights 功能。
- `iam` – 允许 Application Insights 创建服务相关角色，AWSServiceRoleForApplicationInsights。这是必需的，以使 Application Insights 可用执行操作，例如分析客户的资源组、创建 CloudFormation 堆栈以创建有关指标的告警，以及在 EC2 实例上配置 CloudWatch 代理。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "applicationinsights:*",
```

```
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "sqs:ListQueues",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "autoscaling:DescribeAutoScalingGroups",
        "lambda:ListFunctions",
        "dynamodb:ListTables",
        "s3:ListAllMyBuckets",
        "sns:ListTopics",
        "states:ListStateMachines",
        "apigateway:GET",
        "ecs:ListClusters",
        "ecs:DescribeTaskDefinition",
        "ecs:ListServices",
        "ecs:ListTasks",
        "eks:ListClusters",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "logs:DescribeLogGroups",
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
      ],
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "application-insights.amazonaws.com"
```

```
        }
      }
    }
  ]
}
```

AWS 托管式策略：CloudWatchApplicationInsightsReadOnlyAccess

您可以将 `CloudWatchApplicationInsightsReadOnlyAccess` 策略附加到 IAM 身份。

此策略授予管理权限，允许只读访问所有 Application Insights 功能。

权限详细信息

该策略包含以下权限。

- `applicationinsights` – 允许只读访问 Application Insights 功能。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "applicationinsights:Describe*",
                "applicationinsights:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管式策略：CloudwatchApplicationInsightsServiceLinkedRolePolicy

您无法将 CloudwatchApplicationInsightsServiceLinkedRolePolicy 策略附加到 IAM 实体。此策略附加到一个与服务相关的角色，该角色允许 Application Insights 监控客户资源。有关更多信息，请参阅 在 CloudWatch Application Insights 中使用服务相关角色。

对 AWS 托管式策略的 Application Insights 更新

查看有关对 AWS 托管式策略的 Application Insights 更新的详细信息（从该服务开始跟踪这些更改开始）。有关此页面更改的自动提示，请订阅 Application Insights Document history（文档历史记录）页面上的 RSS 源。

更改	描述	日期
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 已添加新权限。 该策略更改允许 Amazon CloudWatch Application Insights 在 CloudFormation 堆栈上启用和禁用终止保护，以管理用于安装和配置 CloudWatch 代理的 SSM 资源。	2024 年 7 月 25 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 增加了列出 CloudFormation 堆栈的新权限。 Amazon CloudWatch Application Insights 需要这些权限来分析和监控嵌套在 CloudFormation 堆栈中的 AWS 资源。	2023 年 4 月 24 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可获取 Amazon VPC 和 Route 53 资源列表的新权限。	2023 年 1 月 23 日

更改	描述	日期
	Amazon CloudWatch Application Insights 需要这些权限，才能使用 Amazon CloudWatch 自动设置最佳实践网络监控。	
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可获取 SSM 命令调用结果的新权限。 Amazon CloudWatch Application Insights 需要这些权限，才能自动检测和监控 Amazon EC2 实例上运行的工作负载。	2022 年 12 月 19 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可描述 Amazon VPC 和 Route 53 资源的新权限。 Amazon CloudWatch Application Insights 需要这些权限，才能读取客户 Amazon VPC 和 Route 53 资源配置，并帮助客户使用 Amazon CloudWatch 自动设置最佳实践网络监控。	2022 年 12 月 19 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可描述 EFS 资源的新权限。 Amazon CloudWatch Application Insights 需要这些权限，才能读取 Amazon EFS 客户资源配置，并帮助客户使用 CloudWatch 自动设置 EFS 监控的最佳实践。	2022 年 10 月 3 日

更改	描述	日期
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可描述 EFS 文件系统的新权限 Amazon CloudWatch Application Insights 需要这些权限，才能通过查询账户中的所有受支持资源来创建基于账户的应用程序。	2022 年 10 月 3 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了可检索 FSx 资源信息的新权限。 Amazon CloudWatch Application Insights 需要这些权限，才能检索有关底层 FSx 卷的充足信息以监控工作负载。	2022 年 9 月 12 日
AWS 托管式策略：CloudWatchApplicationInsightsFullAccess – 更新到现有策略	Application Insights 添加了描述日志组的新权限。 Amazon CloudWatch Application Insights 需要此权限，以确保在创建新应用程序时，账户中具有监控日志组的正确权限。	2022 年 1 月 24 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了创建和删除 CloudWatch Log 订阅筛选器的新权限。 Amazon CloudWatch Application Insights 需要这些权限才能创建订阅筛选器，以便于对已配置应用程序中的资源进行日志监控。	2022 年 1 月 24 日

更改	描述	日期
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了新的权限来描述 Elastic Load Balancer 的目标组和目标健康状况。 Amazon CloudWatch Application Insights 需要这些权限，才能通过查询账户中的所有受支持资源来创建基于账户的应用程序。	2021 年 11 月 4 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了在 Amazon EC2 实例上运行 `AmazonCloudWatch-ManageAgent` SSM 文档的新权限。 Amazon CloudWatch Application Insights 需要此权限才能清除由 Application Insights 创建的 CloudWatch 代理配置文件。	2021 年 9 月 30 日

更改	描述	日期
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了新的权限，以支持基于账户的应用程序监控载入并监控账户中所有受支持的资源。 Amazon CloudWatch Application Insights 需要这些权限才能查询、标记资源并为这些资源创建组。 Application Insights 添加了新的权限以支持对 SNS 主题的监控。 Amazon CloudWatch Application Insights 需要这些权限，才能收集 SNS 资源中的元数据以配置对 SNS 主题的监控。	2021 年 9 月 15 日
AWS 托管式策略：CloudWatchApplicationInsightsFullAccess – 更新到现有策略	Application Insights 添加了描述和列出受支持资源的新权限。 Amazon CloudWatch Application Insights 需要这些权限，才能通过查询账户中的所有受支持资源来创建基于账户的应用程序。	2021 年 9 月 15 日

更改	描述	日期
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了描述 FSx 资源的新权限。 Amazon CloudWatch Application Insights 需要这些权限才能读取客户 FSx 资源配置，并帮助客户使用 CloudWatch 自动设置最佳实践 FSx 监控。	2021 年 8 月 31 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了描述和列出 ECS 和 EKS 服务资源的新权限。 Amazon CloudWatch Application Insights 需要此权限才能读取客户容器资源配置，并帮助客户使用 CloudWatch 自动设置最佳实践容器监控。	2021 年 5 月 18 日
CloudwatchApplicationInsightsServiceLinkedRolePolicy – 对现有策略的更新	Application Insights 添加了新权限，允许 OpsCenter 使用对 `opsitem` 资源类型的资源的 `ssm:AddTagsToResource` 操作来为 OpsItems 贴标签。 OpsCenter 需要此权限。Amazon CloudWatch Application Insights 会创建 OpsItems，以便客户可以使用 AWS SSM OpsCenter 解决问题。	2021 年 4 月 13 日
Athena 开启了跟踪更改	Athena 为其 AWS 托管式策略开启了跟踪更改。	2021 年 4 月 13 日

Amazon CloudWatch 权限参考

下表列出每个 CloudWatch API 操作以及您可授权执行该操作的相应操作。可在策略的 `Action` 字段中指定操作，在策略的 `Resource` 字段中指定通配符 (*) 作为资源值。

您可以在 CloudWatch 策略中使用 AWS 范围的条件键来表达条件。有关 AWS 范围的键的完整列表，请参阅 IAM 用户指南中的 AWS 全局和 IAM 条件上下文键。

> **Note**
>
> 要指定操作，请在 API 操作名称之前使用 `cloudwatch:` 前缀。例如：`cloudwatch:GetMetricData`、`cloudwatch:ListMetrics` 或 `cloudwatch:*`（适用于所有 CloudWatch 操作）。

主题

- CloudWatch API 操作和必需的操作权限
- CloudWatch Contributor Insights API 操作和所需的操作权限
- CloudWatch Events API 操作和所需的操作权限
- CloudWatch Logs API 操作和所需的操作权限
- Amazon EC2 API 操作和所需的操作权限
- Amazon EC2 Auto Scaling API 操作和所需的操作权限

CloudWatch API 操作和必需的操作权限

CloudWatch API 操作	所需权限（API 操作）
DeleteAlarms	`cloudwatch:DeleteAlarms` 要求删除警报。
DeleteDashboards	`cloudwatch:DeleteDashboards` 删除控制面板所必需。

CloudWatch API 操作	所需权限（API 操作）
DeleteMetricStream	`cloudwatch:DeleteMetricStream` 删除指标流所需。
DescribeAlarmHistory	`cloudwatch:DescribeAlarmHistory` 要求查看警报历史记录。要检索有关复合告警的信息，`cloudwatch:DescribeAlarmHistory` 权限必须具有 * 范围。如果您的 `cloudwatch:DescribeAlarmHistory` 权限的范围较窄，则无法返回有关复合告警的信息。
DescribeAlarms	`cloudwatch:DescribeAlarms` 检索有关告警的信息所需。 要检索有关复合告警的信息，`cloudwatch:DescribeAlarms` 权限必须具有 * 范围。如果您的 `cloudwatch:DescribeAlarms` 权限的范围较窄，则无法返回有关复合告警的信息。
DescribeAlarmsForMetric	`cloudwatch:DescribeAlarmsForMetric` 要求查看指标的警报。
DisableAlarmActions	`cloudwatch:DisableAlarmActions` 要求禁用警报操作。
EnableAlarmActions	`cloudwatch:EnableAlarmActions` 要求启用警报操作。

CloudWatch API 操作	所需权限（API 操作）
GetDashboard	`cloudwatch:GetDashboard` 若要显示有关现有控制面板的数据，则是必需的。
GetMetricData	`cloudwatch:GetMetricData` 在 CloudWatch 控制台中检索大量指标数据以及对该数据执行指标数学运算所需。
GetMetricStatistics	`cloudwatch:GetMetricStatistics` 在 CloudWatch 控制台的其他部分和控制面板小部件中查看图表所需。
GetMetricStream	`cloudwatch:GetMetricStream` 查看指标流信息所需。
GetMetricWidgetImage	`cloudwatch:GetMetricWidgetImage` 将一个或多个 CloudWatch 指标的图表快照作为位图图像检索所需。
ListDashboards	`cloudwatch:ListDashboards` 查看您的账户中 CloudWatch 控制面板的列表所需。
ListMetrics	`cloudwatch:ListMetrics` 在 CloudWatch 控制台和 CLI 中查看或搜索指标名称所需。要求在控制面板小部件上选择指标。

CloudWatch API 操作	所需权限（API 操作）
ListMetricStreams	`cloudwatch:ListMetricStreams` 查看或搜索账户中指标流列表所需。
PutCompositeAlarm	`cloudwatch:PutCompositeAlarm` 创建复合告警所需 要创建复合告警，`cloudwatch:PutCompositeAlarm` 权限必须具有 * 范围。如果您的 `cloudwatch:PutCompositeAlarm` 权限的范围较窄，则无法返回有关复合告警的信息。
PutDashboard	`cloudwatch:PutDashboard` 创建控制面板或更新现有控制面板所必需。
PutMetricAlarm	`cloudwatch:PutMetricAlarm` 要求创建或更新警报。
PutMetricData	`cloudwatch:PutMetricData` 若要创建指标，则是必需的。
PutMetricStream	`cloudwatch:PutMetricStream` 创建指标流所需
SetAlarmState	`cloudwatch:SetAlarmState` 要求手动设置警报的状态。

CloudWatch API 操作	所需权限（API 操作）
StartMetricStreams	`cloudwatch:StartMetricStreams` 开启指标流中的指标流程所需。
StopMetricStreams	`cloudwatch:StopMetricStreams` 临时停止指标流中的指标流程所需。
TagResource	`cloudwatch:TagResource` 在 CloudWatch 资源（如告警和 Contributor Insights 规则）上添加或更新标签所需。
UntagResource	`cloudwatch:UntagResource` 从 CloudWatch 资源中移除标签所需。

CloudWatch Contributor Insights API 操作和所需的操作权限

Important

当您向用户授予 `cloudwatch:PutInsightRule` 权限时，默认情况下，该用户可以创建一个规则来评估 CloudWatch Logs 中的任何日志组。您可以添加 IAM 策略条件，以限制用户的这些权限，使其包含和排除特定的日志组。有关更多信息，请参阅 使用条件键限制 Contributor Insights 用户对日志组的访问。

CloudWatch Contributor Insights API 操作	所需权限（API 操作）
DeleteInsightRules	`cloudwatch:DeleteInsightRules` 删除 Contributor Insights 规则所需。

CloudWatch Contributor Insights API 操作	所需权限（API 操作）
DescribeInsightRules	`cloudwatch:DescribeInsightRules` 查看您账户中的 Contributor Insights 规则所需。
EnableInsightRules	`cloudwatch:EnableInsightRules` 启用 Contributor Insights 规则所需。
GetInsightRuleReport	`cloudwatch:GetInsightRuleReport` 检索 Contributor Insights 规则收集的时间序列数据和其他统计数据所需。
PutInsightRule	`cloudwatch:PutInsightRule` 创建 Contributor Insights 规则所需。请参阅此表开头的 Important（重要提示）信息。

CloudWatch Events API 操作和所需的操作权限

CloudWatch Events API 操作	所需权限（API 操作）
DeleteRule	`events:DeleteRule` 删除规则所必需的。
DescribeRule	`events:DescribeRule` 列出有关规则的详细信息所必需的。
DisableRule	`events:DisableRule`

CloudWatch Events API 操作	所需权限（API 操作）
	禁用规则所必需的。
EnableRule	events:EnableRule 启用规则所必需的。
ListRuleNamesByTarget	events:ListRuleNamesByTarget 列出与目标关联的规则所必需的。
ListRules	events:ListRules 列出您账户中的所有规则所必需的。
ListTargetsByRule	events:ListTargetsByRule 列出与规则关联的所有目标所必需的。
PutEvents	events:PutEvents 添加可匹配到规则的自定义活动所必需的。
PutRule	events:PutRule 创建或更新规则所必需的。
PutTargets	events:PutTargets 将目标添加到规则所必需的。
RemoveTargets	events:RemoveTargets 从规则中删除目标所必需的。

CloudWatch Events API 操作	所需权限（API 操作）
TestEventPattern	`events:TestEventPattern` 针对给定事件测试事件模式所必需的。

CloudWatch Logs API 操作和所需的操作权限

CloudWatch Logs API 操作	所需权限（API 操作）
CancelExportTask	`logs:CancelExportTask` 需要取消待处理或正在运行的导出任务。
CreateExportTask	`logs:CreateExportTask` 将数据从日志组导出到 Amazon S3 存储桶所需。
CreateLogGroup	`logs:CreateLogGroup` 需要创建新的日志组。
CreateLogStream	`logs:CreateLogStream` 需要在日志组中创建新的日志流。
DeleteDestination	`logs:DeleteDestination` 需要删除日志目标并对其禁用所有订阅筛选器。
DeleteLogGroup	`logs:DeleteLogGroup` 需要删除日志组和所有关联的存档日志事件。

CloudWatch Logs API 操作	所需权限（API 操作）
DeleteLogStream	`logs:DeleteLogStream` 需要删除日志流和所有关联的存档日志事件。
DeleteMetricFilter	`logs:DeleteMetricFilter` 需要删除与日志组关联的指标筛选器。
DeleteQueryDefinition	`logs:DeleteQueryDefinition` 删除 CloudWatch Logs Insights 中保存的查询定义所需。
DeleteResourcePolicy	`logs:DeleteResourcePolicy` 删除 CloudWatch Logs 资源策略所需。
DeleteRetentionPolicy	`logs:DeleteRetentionPolicy` 需要删除日志组的保留策略。
DeleteSubscriptionFilter	`logs:DeleteSubscriptionFilter` 需要删除与日志组关联的订阅筛选器。
DescribeDestinations	`logs:DescribeDestinations` 需要查看与账户关联的所有目标。
DescribeExportTasks	`logs:DescribeExportTasks` 需要查看与账户关联的所有导出任务。

CloudWatch Logs API 操作	所需权限（API 操作）
DescribeLogGroups	`logs:DescribeLogGroups` 需要查看与账户关联的所有日志组。
DescribeLogStreams	`logs:DescribeLogStreams` 需要查看与日志组关联的所有日志流。
DescribeMetricFilters	`logs:DescribeMetricFilters` 需要查看与日志组关联的所有指标。
DescribeQueryDefinitions	`logs:DescribeQueryDefinitions` 查看 CloudWatch Logs Insights 中保存的查询定义列表所需。
DescribeQueries	`logs:DescribeQueries` 查看已计划、正在执行或最近已执行的 CloudWatch Logs Insights 查询列表所需。
DescribeResourcePolicies	`logs:DescribeResourcePolicies` 查看 CloudWatch Logs 资源策略列表所需。
DescribeSubscriptionFilters	`logs:DescribeSubscriptionFilters` 需要查看与日志组关联的所有订阅筛选器。

CloudWatch Logs API 操作	所需权限（API 操作）
FilterLogEvents	`logs:FilterLogEvents` 需要按照日志组筛选器模式对日志事件进行排序。
GetLogEvents	`logs:GetLogEvents` 需要从日志流中检索日志事件。
GetLogGroupFields	`logs:GetLogGroupFields` 需要检索日志组中日志事件内包含的字段列表。
GetLogRecord	`logs:GetLogRecord` 需要从单个日志事件中检索详细信息。
GetQueryResults	`logs:GetQueryResults` 检索 CloudWatch Logs Insights 查询的结果所需。
ListTagsLogGroup	`logs:ListTagsLogGroup` 需要列出与日志组关联的标签。
PutDestination	`logs:PutDestination` 创建或更新目标日志流（例如，Kinesis 流）所需。

CloudWatch Logs API 操作	所需权限（API 操作）
PutDestinationPolicy	`logs:PutDestinationPolicy` 需要创建或更新与现有日志目标关联的访问策略。
PutLogEvents	`logs:PutLogEvents` 需要将一批日志事件上传到日志流。
PutMetricFilter	`logs:PutMetricFilter` 需要创建或更新指标筛选器并将其与日志组关联。
PutQueryDefinition	`logs:PutQueryDefinition` 在 CloudWatch Logs Insights 中保存查询所需。
PutResourcePolicy	`logs:PutResourcePolicy` 创建 CloudWatch Logs 资源策略所需。
PutRetentionPolicy	`logs:PutRetentionPolicy` 需要设置日志组中的日志事件的保存（保留）天数。
PutSubscriptionFilter	`logs:PutSubscriptionFilter` 需要创建或更新订阅筛选器并将其与日志组关联。

CloudWatch Logs API 操作	所需权限（API 操作）
StartQuery	`logs:StartQuery` 启动 CloudWatch Logs Insights 查询所需。
StopQuery	`logs:StopQuery` 停止正在执行的 CloudWatch Logs Insights 查询所需。
TagLogGroup	`logs:TagLogGroup` 需要添加或更新日志组标签。
TestMetricFilter	`logs:TestMetricFilter` 需要针对日志事件消息采样测试筛选器模式。

Amazon EC2 API 操作和所需的操作权限

Amazon EC2 API 操作	所需权限（API 操作）
DescribeInstanceStatus	`ec2:DescribeInstanceStatus` 查看 EC2 实例状态详细信息所必需的。
DescribeInstances	`ec2:DescribeInstances` 查看 EC2 实例详细信息所必需的。
RebootInstances	`ec2:RebootInstances` 重启 EC2 实例所必需的。

Amazon EC2 API 操作	所需权限（API 操作）
StopInstances	`ec2:StopInstances` 停止 EC2 实例所必需的。
TerminateInstances	`ec2:TerminateInstances` 终止 EC2 实例所必需的。

Amazon EC2 Auto Scaling API 操作和所需的操作权限

Amazon EC2 Auto Scaling API 操作	所需权限（API 操作）
扩展	`autoscaling:Scaling` 扩展 Auto Scaling 组所需。
触发器	`autoscaling:Trigger` 触发 Auto Scaling 操作所需。

Amazon CloudWatch 的合规性验证

作为多项 AWS 合规性计划的一部分，第三方审计员将评估 Amazon CloudWatch 的安全性和合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其它。

有关特定合规性计划范围内的 AWS 服务列表，请参阅合规性计划范围内的 AWS 服务。有关常规信息，请参阅AWS合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅下载 AWS Artifact 中的报告。

您使用 Amazon CloudWatch 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用法律法规。AWS 提供以下资源来帮助您满足合规性：

- 安全性与合规性 Quick Start 指南 - 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署基于安全性和合规性的基准环境的步骤。

- 《设计符合 HIPAA 安全性和合规性要求的架构》白皮书 – 此白皮书介绍公司如何使用AWS创建符合HIPAA 标准的应用程序。
- AWS合规性资源 – 此业务手册和指南集合可能适用于您的行业和位置。
- 《AWS Config 开发人员指南》中的使用规则评估资源 – AWS Config；评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub – 此AWS服务提供了AWS中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实践。

Amazon CloudWatch 中的恢复能力

AWS全球基础设施围绕AWS区域和可用区构建。区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关AWS区域和可用区的更多信息，请参阅AWS全球基础设施。

Amazon CloudWatch 中的基础设施安全性

作为一项托管式服务，Amazon CloudWatch 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础设施的信息，请参阅 AWS 云安全。要按照基础架构安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础架构保护。

您可以使用 AWS 发布的 API 调用，来通过网络访问 CloudWatch。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

网络隔离

Virtual Private Cloud (VPC) 是 Amazon Web Services 云内您自己的逻辑隔离区域中的虚拟网络。子网是 VPC 中的 IP 地址范围。您可以在 VPC 的子网中部署各种 AWS 资源。例如，可以在子网中部署 Amazon EC2 实例、EMR 集群和 DynamoDB 表。有关更多信息，请参阅 Amazon VPC 用户指南。

要使 CloudWatch 能够在不访问公有 Internet 的情况下与 VPC 中的资源进行通信，请使用 AWS PrivateLink。有关更多信息，请参阅 将 CloudWatch 和 CloudWatch Synthetics 与接口 VPC 终端节点结合使用。

私有子网是不具有到公共 Internet 的默认路由的子网。在私有子网中部署 AWS 资源不会阻止 Amazon CloudWatch 从资源中收集内置指标。

如果您需要在私有子网发布来自 AWS 资源的自定义指标，则可使用代理服务器执行此操作。代理服务器会将这些 HTTPS 请求转发到 CloudWatch 的公有 API 端点。

AWS Security Hub

监控 CloudWatch 的使用情况，因为它与使用 AWS Security Hub 的安全最佳实践有关。Security Hub 使用安全控件来评估资源配置和安全标准，以帮助您遵守各种合规框架。有关使用 Security Hub 评估 CloudWatch 资源的更多信息，请参阅《AWS Security Hub 用户指南》中的 Amazon CloudWatch 控件。

将 CloudWatch 和 CloudWatch Synthetics 与接口 VPC 终端节点结合使用

如果您使用 Amazon Virtual Private Cloud (Amazon VPC) 托管 AWS 资源，则可以在您的 VPC、CloudWatch 和 CloudWatch Synthetics 之间建立私有连接。您可以使用此连接实现 CloudWatch 和 CloudWatch Synthetics 与您的 VPC 上的资源的通信而不用访问公共互联网。

Amazon VPC 是一项 AWS 服务，可用来启动在虚拟网络中定义的 AWS 资源。借助 VPC，您可以控制您的网络设置，如 IP 地址范围、子网、路由表和网络网关。要将您的 VPC 连接到 CloudWatch 或 CloudWatch Synthetics，您需要定义一个接口 VPC 终端节点以将您的 VPC 连接到 AWS 服务。该端点提供了到 CloudWatch 或 CloudWatch Synthetics 的可靠、可扩展的连接，无需互联网网关、网络地址转换 (NAT) 实例或 VPN 连接。有关更多信息，请参阅 Amazon VPC 用户指南中的什么是 Amazon VPC。

接口 VPC 终端节点由 AWS PrivateLink 提供支持，后者是一种AWS技术，可将弹性网络接口与私有 IP 地址结合使用来支持AWS服务之间的私有通信。有关更多信息，请参阅新增 – 适用于 AWS 服务的 AWS PrivateLink 博客文章。

以下步骤适用于 Amazon VPC 的用户。有关更多信息，请参阅 Amazon VPC 用户指南中的开始使用。

CloudWatch VPC 终端节点

CloudWatch 当前在以下 AWS 区域中支持 VPC 终端节点：

- 美国东部（俄亥俄州）
- 美国东部（弗吉尼亚州北部）
- 美国西部（北加利福尼亚）
- 美国西部（俄勒冈州）
- 亚太地区（香港）
- 亚太地区（孟买）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲（伦敦）
- 欧洲地区（巴黎）
- 中东（阿联酋）
- 南美洲（圣保罗）
- AWS GovCloud（美国东部）
- AWS GovCloud（美国西部）

为 CloudWatch 创建 VPC 终端节点

要开始使用 CloudWatch 与 VPC，请为 CloudWatch 创建一个接口 VPC 终端节点。要选择的服务名称为 `com.amazonaws.region.monitoring`。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口端点。

您不需要更改 CloudWatch 的设置。CloudWatch 使用公有端点或私有接口 VPC 终端节点（二者中在使用中的那个）调用其他 AWS 服务。例如，如果为 CloudWatch 创建接口 VPC 终端节点，并且您已拥有从位于 VPC 上的资源流向 CloudWatch 的指标，默认情况下，这些指标将开始流过接口 VPC 终端节点。

控制对 CloudWatch VPC 终端节点的访问

VPC 终端节点策略是一种 IAM 资源策略，您在创建或修改端点时可将它附加到端点。如果您在创建端点时未附加策略，Amazon VPC 会为您附加一个默认策略，该策略允许对服务的完全访问。端点策略不会覆盖或替换用户策略或服务特定的策略。这是一个单独的策略，用于控制从终端节点中对指定服务进行的访问。

终端节点策略必须采用 JSON 格式编写。

有关更多信息，请参阅《Amazon VPC 用户指南》中的 使用 VPC 端点控制对服务的访问。

下面是用于 CloudWatch 的端点策略示例。该策略允许通过 VPC 连接到 CloudWatch 的用户将指标数据发送到 CloudWatch，并禁止他们执行其他 CloudWatch 操作。

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

编辑 CloudWatch 的 VPC 终端节点策略

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择 Endpoints（端点）。
3. 如果还没有为 CloudWatch 创建端点，请选择 Create Endpoint（创建端点）。接下来，选择 com.amazonaws.***region***.monitoring，然后选择 Create Endpoint (创建终端节点)。
4. 选择 com.amazonaws.***region***.monitoring 终端节点，然后选择 Policy (策略) 选项卡。
5. 选择 Edit Policy (编辑策略)，然后进行更改。

CloudWatch Synthetics VPC 终端节点

CloudWatch Synthetics 当前在以下 AWS 区域中支持 VPC 终端节点：

- 美国东部（俄亥俄州）
- 美国东部（弗吉尼亚州北部）
- 美国西部（北加利福尼亚）
- 美国西部（俄勒冈州）
- 亚太地区（香港）
- 亚太地区（孟买）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲（伦敦）
- 欧洲（巴黎）
- 南美洲（圣保罗）

为 CloudWatch Synthetics 创建 VPC 终端节点

要开始将 CloudWatch Synthetics 与您的 VPC 结合使用，请为 CloudWatch Synthetics 创建一个接口 VPC 终端节点。要选择的服务名称为 `com.amazonaws.region.synthetics`。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口端点。

您不需要更改 CloudWatch Synthetics 的设置。CloudWatch Synthetics 使用公有终端节点或私有接口 VPC 终端节点（二者中在使用的那个）与其他 AWS 服务通信。例如，如果您为 CloudWatch Synthetics 创建了接口 VPC 终端节点，并且您已有 Amazon S3 的接口端点，则默认情况下，CloudWatch Synthetics 通过接口 VPC 终端节点开始与 Amazon S3 通信。

控制对您的 CloudWatch Synthetics VPC 终端节点的访问

VPC 终端节点策略是一种 IAM 资源策略，您在创建或修改端点时可将它附加到端点。如果在创建端点时未附加策略，我们将为您附加默认策略以允许对服务进行完全访问。端点策略不会覆盖或替换用户策略或服务特定的策略。这是一个单独的策略，用于控制从终端节点中对指定服务进行的访问。

端点策略会影响由 VPC 私有托管的 canary。在私有子网上运行的 Canary 不需要这些策略。

终端节点策略必须采用 JSON 格式编写。

有关更多信息，请参阅《Amazon VPC 用户指南》中的 使用 VPC 端点控制对服务的访问。

以下是 CloudWatch Synthetics 的端点策略示例。此策略允许通过 VPC 连接到 CloudWatch Synthetics 的用户查看有关 Canary 及其运行的信息，但不能创建、修改或删除 Canary。

```
{
    "Statement": [
        {
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:GetCanaryRuns"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

编辑 CloudWatch Synthetics 的 VPC 终端节点策略

1. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择 Endpoints（端点）。
3. 如果还没有为 CloudWatch Synthetics 创建端点，请选择 Create Endpoint（创建端点）。选择 com.amazonaws.***region***.synthetics，然后选择 Create endpoint (创建终端节点)。
4. 选择 com.amazonaws.***region***.synthetics 终端节点，然后选择 Policy (策略) 选项卡。
5. 选择 Edit Policy (编辑策略)，然后进行更改。

Synthetics 金丝雀的安全注意事项

以下部分说明了在 Synthetics 中创建和运行金丝雀时应考虑的安全问题。

使用安全连接

由于金丝雀代码和金丝雀测试运行的结果可能包含敏感信息，因此，请不要让金丝雀通过未加密的连接来连接到终端节点。始终使用加密连接，例如以 `https://` 开头的连接。

金丝雀命名注意事项

金丝雀的 Amazon Resource Name (ARN) 包含在用户代理标头中，作为从 CloudWatch Synthetics 包装器库中包含的 Puppeteer 驱动的 Chromium 浏览器发出的出站调用的一部分。这有助于标识 CloudWatch Synthetics 金丝雀流量，并再次将其与正在发出调用的金丝雀关联。

金丝雀 ARN 包含金丝雀名称。选择未显示专有信息的金丝雀名称。

此外，请确保仅将您的金丝雀指向由您控制的网站和终端节点。

金丝雀代码中的密钥和敏感信息

如果您使用 zip 文件将金丝雀代码直接传递到金丝雀中，则该脚本的内容会显示在 AWS CloudTrail 日志中。

如果您在金丝雀脚本中有敏感信息或密钥（例如访问密钥或数据库凭证），我们强烈建议您将该脚本作为版本控制对象存储在 Amazon S3 中，并将 Amazon S3 位置传递到金丝雀中，而不是通过 zip 文件传递金丝雀代码。

如果您确实要使用 zip 文件传递金丝雀脚本，我们强烈建议您不要将密钥或敏感信息包含在金丝雀源代码中。有关如何使用 AWS Secrets Manager 来帮助保护密钥安全的更多信息，请参阅什么是 AWS Secrets Manager？。

权限注意事项

我们建议您限制对由 CloudWatch Synthetics 创建或使用的资源的访问。对金丝雀存储测试运行结果和其他构件（例如，日志和屏幕截图）的 Amazon S3 存储桶使用严格权限。

同样，对金丝雀源代码的存储位置保留严格权限，以免用户意外或恶意删除用于金丝雀的 Lambda 层或 Lambda 函数。

要帮助确保您运行所需的金丝雀代码，您可以对用于存储金丝雀代码的 Amazon S3 存储桶使用对象版本控制。之后，在您指定此代码作为金丝雀运行时，您可以将对象 `versionId` 作为路径的一部分，如以下示例所示。

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

堆栈跟踪和异常消息

预设情况下，CloudWatch Synthetics 金丝雀会捕获金丝雀脚本引发的所有异常，无论该脚本是自定义脚本还是蓝图中的脚本。CloudWatch Synthetics 将异常消息和堆栈跟踪记录到三个位置：

- 记录到 CloudWatch Synthetics 服务中，以便在描述测试运行时加快调试速度
- 记录到 CloudWatch Logs 中，具体取决于创建 Lambda 函数时使用的配置
- 记录到 Synthetics 日志文件中，该文件是一个纯文本文件，已上载到由您为金丝雀的 `resultsLocation` 设置的值所指定的 Amazon S3 位置

如果您要发送和存储的信息较少，则可以在异常返回到 CloudWatch Synthetics 包装器库之前捕获异常。

您也可以将请求 URL 包含在错误中。CloudWatch Synthetics 会扫描脚本引发的错误中的任何 URL，并根据 restrictedUrlParameters 配置编辑其中的受限 URL 参数。如果您在脚本中记录错误消息，则可以使用 getSanitizedErrorMessage 在记录日志之前编辑 URL。

缩小 IAM 角色的范围

建议您不要将金丝雀配置为访问潜在的恶意 URL 或终端节点。将金丝雀指向不受信任或未知的网站或端点可能会将您的 Lambda 函数代码暴露给恶意用户的脚本。假设一个恶意网站可以破解 Chromium，它可能会以与您使用 Internet 浏览器连接时类似的方式访问您的 Lambda 代码。

使用缩小了权限范围的 IAM 执行角色来运行您的 Lambda 函数。这样，如果 Lambda 函数受到恶意脚本的影响，它在作为金丝雀的 AWS 账户运行时可采取的操作会受到限制。

在使用 CloudWatch 控制台创建金丝雀时，会使用缩小了权限范围的 IAM 执行角色来进行创建。

敏感数据编辑

CloudWatch Synthetics 捕获 URL、状态代码、故障原因（如有）以及请求和响应的标头及请求体和响应体。这使得金丝雀用户能够理解、监控和调试金丝雀。

以下各节中所述配置可以在金丝雀执行的任何时间点进行设置。您还可以选择对不同的 Synthetics 步骤应用不同的配置。

请求 URL

预设情况下，CloudWatch Synthetics 会将每个 URL 的请求 URL、状态代码和状态原因记录在金丝雀日志中。请求 URL 也可以出现在金丝雀执行报告、HAR 文件等中。请求 URL 可能会包含敏感查询参数，例如访问令牌或密码。您可以编辑敏感信息，以免被 CloudWatch Synthetics 记录下来。

若要修改敏感信息，请设置配置属性 restrictedUrlParameters。有关更多信息，请参阅SyntheticsConfiguration 类。配置此属性后，CloudWatch Synthetics 会在记录日志前根据 restrictedUrlParameters 属性修改 URL 参数，包括路径和查询参数值。如果您在脚本中记录 URL，则可以使用 getSanitizedUrl(url, stepConfig = null) 在记录日志之前编辑 URL。有关更多信息，请参阅SyntheticsLogHelper 类。

标头

预设情况下，CloudWatch Synthetics 不记录请求/响应标头。对于 UI 金丝雀，此为使用 `syn-nodejs-puppeteer-3.2` 和更高版本运行时金丝雀的默认行为。

如果标头不包含敏感信息，则可以通过将 includeRequestHeaders 和 includeResponseHeaders 属性设置为 `true` 来在 HAR 文件和 HTTP 报告中启用标头。您可以启用所有标头，但选择限制敏感标头键的值。例如，您可以选择只编辑金丝雀生成的构件中的 `Authorization` 标头。

请求体和响应体

预设情况下，CloudWatch Synthetics 不会将请求/响应体记录在金丝雀日志或报告中。此信息对 API 金丝雀特别有用。Synthetics 捕获所有 HTTP 请求，并可显示标头、请求体和响应体。有关更多信息，请参阅executeHttpStep(stepName, requestOptions, [callback], [stepConfig])。您可以选择将 includeRequestBody 和 includeResponseBody 属性设置为 `true` 来启用请求/响应体。

使用 AWS CloudTrail 记录 Amazon CloudWatch API 和控制台操作

Amazon CloudWatch、CloudWatch Synthetics、CloudWatch RUM 和 CloudWatch Internet Montitor 与 AWS CloudTrail 集成，后者是一种提供用户、角色或 AWS 服务所采取操作的记录的服务。CloudTrail 捕获由您的 AWS 账户发出或代表该账户发出的 API 调用。捕获的调用包含来自控制台的调用和对 API 操作的代码调用。

如果您创建了一个跟踪记录，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 CloudWatch 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台中的 事件历史记录 中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 CloudWatch 发出的请求内容、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息（包括如何对其进行配置和启用），请参阅 AWS CloudTrail 用户指南。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management（IAM）用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

要持续记录 AWS 账户中的事件（包括 CloudWatch 和 CloudWatch Synthetics 的事件），请创建跟踪记录。通过跟踪，CloudTrail 可将日志文件传送至 S3 存储桶。默认情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 S3 存储桶。您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取措施。有关更多信息，请参阅下列内容：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

> **Note**
>
> 有关 CloudTrail 中记录的 CloudWatch Logs API 调用的信息，请参阅 CloudTrail 中的 CloudWatch Logs 信息。

主题

- CloudTrail 中的 CloudWatch 信息
- CloudTrail 中的 CloudWatch 数据面板事件
- CloudTrail 中的查询生成信息
- CloudTrail 中的 CloudWatch Internet Monitor
- CloudTrail 中的 CloudWatch Synthetics 信息
- CloudTrail 中的 CloudWatch RUM 信息
- CloudTrail 中的 CloudWatch RUM 数据面板事件

CloudTrail 中的 CloudWatch 信息

CloudWatch 支持将以下操作记录为 CloudTrail 日志文件中的事件：

- DeleteAlarms
- DeleteAnomalyDetector
- DeleteDashboards
- DescribeAlarmHistory
- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- DisableAlarmActions
- EnableAlarmActions
- GetDashboard
- ListDashboards
- PutAnomalyDetector
- PutDashboard

- PutMetricAlarm
- SetAlarmState

示例：CloudWatch 日志文件条目

下面的示例显示了一个 CloudTrail 日志条目，该条目演示了 PutMetricAlarm 操作。


```
{
    "Records": [{
        "eventVersion": "1.01",
        "userIdentity": {
            "type": "Root",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID"
        },
        "eventTime": "2014-03-23T21:50:34Z",
        "eventSource": "monitoring.amazonaws.com",
        "eventName": "PutMetricAlarm",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "threshold": 50.0,
            "period": 60,
            "metricName": "CloudTrail Test",
            "evaluationPeriods": 3,
            "comparisonOperator": "GreaterThanThreshold",
            "namespace": "AWS/CloudWatch",
            "alarmName": "CloudTrail Test Alarm",
            "statistic": "Sum"
        },
        "responseElements": null,
        "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
        "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
    },
    ..additional entries
  ]
  }
```


以下日志文件条目显示某个用户调用了 CloudWatch Events PutRule 操作。

```
{
         "eventVersion":"1.03",
         "userIdentity":{
            "type":"Root",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:root",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext":{
               "attributes":{
                  "mfaAuthenticated":"false",
                  "creationDate":"2015-11-17T23:56:15Z"
               }
            }
         },
         "eventTime":"2015-11-18T00:11:28Z",
         "eventSource":"events.amazonaws.com",
         "eventName":"PutRule",
         "awsRegion":"us-east-1",
         "sourceIPAddress":"AWS Internal",
         "userAgent":"AWS CloudWatch Console",
         "requestParameters":{
            "description":"",
            "name":"cttest2",
            "state":"ENABLED",
            "eventPattern":"{\"source\":[\"aws.ec2\"],\"detail-type\":[\"EC2 Instance
 State-change Notification\"]}",
            "scheduleExpression":""
         },
         "responseElements":{
            "ruleArn":"arn:aws:events:us-east-1:123456789012:rule/cttest2"
         },
         "requestID":"e9caf887-8d88-11e5-a331-3332aa445952",
         "eventID":"49d14f36-6450-44a5-a501-b0fdcdfaeb98",
         "eventType":"AwsApiCall",
         "apiVersion":"2015-10-07",
         "recipientAccountId":"123456789012"
}
```

以下日志文件条目显示某个用户调用了 CloudWatch Logs `CreateExportTask` 操作。

```
{
        "eventVersion": "1.03",
```

```
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/someuser",
            "accountId": "123456789012",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "someuser"
        },
        "eventTime": "2016-02-08T06:35:14Z",
        "eventSource": "logs.amazonaws.com",
        "eventName": "CreateExportTask",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "destination": "yourdestination",
            "logGroupName": "yourloggroup",
            "to": 123456789012,
            "from": 0,
            "taskName": "yourtask"
        },
        "responseElements": {
            "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
        },
        "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
        "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
        "eventType": "AwsApiCall",
        "apiVersion": "20140328",
        "recipientAccountId": "123456789012"
}
```

CloudTrail 中的 CloudWatch 数据面板事件

CloudTrail 可以捕获与 CloudWatch 数据面板操作 GetMetricData 和 GetMetricWidgetImage 相关的 API 活动。

数据事件也称为数据面板操作，可让您深入了解在资源上或资源内执行的资源操作。数据事件通常是高容量活动。

要在 CloudTrail 文件中启用 GetMetricData 和 GetMetricWidgetImage 数据事件的日志记录，需要在 CloudTrail 中启用数据面板 API 活动的日志记录。有关更多信息，请参阅记录数据事件用于跟踪。

可以按资源类型筛选数据面板事件。由于在 CloudTrail 中使用数据事件会产生额外费用，因此按资源筛选可以让您更好地控制自己记录和支付费用的内容。

使用 CloudTrail 收集的信息，可以确定向 CloudWatch GetMetricData 或 GetMetricWidgetImage API 发出的特定请求、请求者的 IP 地址、请求者的身份以及请求的日期和时间。使用 CloudTrail 记录 GetMetricData 和 GetMetricWidgetImage API 可帮助您实现 AWS 账户的运营和风险审计、治理与合规性。

> **Note**
>
> 当您在 CloudTrail 中查看 GetMetricData 事件时，可能会看到比您发起的调用更多的调用。这是因为 CloudWatch 将由内部组件发起的 GetMetricData 操作的事件记录到 CloudTrail。例如，在跨账户可观测性中，您将看到 CloudWatch 控制面板发起 GetMetricData 调用以刷新小组件数据，而监控账户发起 GetMetricData 调用以从源账户检索数据。这些内部发起的调用不会产生 CloudWatch 费用，但会计入 CloudTrail 中记录的事件数量，而 CloudTrail 根据记录的事件数量收费。

以下是 GetMetricData 操作的 CloudTrail 事件示例。


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDA2NYTR2EPCTNY7AF3L",
        "arn": "arn:aws:iam::111122223333:user/admin",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE1234567890",
        "userName": "admin"
    },
    "eventTime": "2024-05-08T16:20:34Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GetMetricData",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "99.45.3.7",
    "userAgent": "aws-cli/2.13.23 Python/3.11.5 Darwin/23.4.0 exe/x86_64 prompt/off
 command/cloudwatch.get-metric-data",
    "requestParameters": {
        "metricDataQueries": [{
                "id": "e1",
                "expression": "m1 / m2",
```

```
                "label": "ErrorRate"
            },
            {
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum",
                    "unit": "Count"
                },
                "returnData": false
            },
            {
                "id": "m2",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum"
                },
                "returnData": true
            }
        ],
        "startTime": "Apr 19, 2024, 4:00:00 AM",
        "endTime": "May 8, 2024, 4:30:00 AM"
    },
    "responseElements": null,
    "requestID": "EXAMPLE-57ac-47d5-938c-f5917c6799d5",
    "eventID": "EXAMPLE-211c-404b-b13d-36d93c8b4fbf",
    "readOnly": true,
    "resources": [{
```

```
        "type": "AWS::CloudWatch::Metric"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
}
```

CloudTrail 中的查询生成信息

还支持查询生成器控制台事件的 CloudTrail 日志记录。查询生成器目前支持 CloudWatch Metric Insights 和 CloudWatch Logs Insights。在这些 CloudTrail 事件中，`eventSource` 为 `monitoring.amazonaws.com`。

以下示例显示了一个 CloudTrail 日志条目，该条目演示了 CloudWatch Metrics Insights 中的 GenerateQuery 操作。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2020-04-08T21:43:24Z",
                "mfaAuthenticated": "false"
            }
```

```
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GenerateQuery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "exampleUserAgent",
    "requestParameters": {
        "query_ask": "***",
        "query_type": "MetricsInsights",
        "metrics_insights": {
            "aws_namespaces": [
                "AWS/S3",
                "AWS/Lambda",
                "AWS/DynamoDB"
            ]
        },
        "include_description": true
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


CloudTrail 中的 CloudWatch Internet Monitor

CloudWatch 互联网监测仪支持将以下操作记录为 CloudTrail 日志文件中的事件。

- CreateMonitor
- DeleteMonitor
- GetHealthEvent
- GetMonitor
- GetQueryResults
- GetQueryStatus

- ListHealthEvents
- ListMonitors
- ListTagsForResource
- StartQuery
- StopQuery
- UpdateMonitor

示例：CloudWatch Internet Monitor 日志文件条目

以下示例显示了一个 CloudTrail Internet Monitor 日志条目，该条目演示了 `ListMonitors` 操作。


```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
```

```
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```

以下示例显示了一个 CloudTrail Internet Monitor 日志条目，该条目演示了 `CreateMonitor` 操作。

```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:08Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "CreateMonitor",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": {
```

```
            "MonitorName": "TestMonitor",
            "Resources": ["arn:aws:ec2:us-east-2:444455556666:vpc/vpc-febc0b95"],
            "ClientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": {
            "Arn": "arn:aws:internetmonitor:us-east-2:444455556666:monitor/ct-
onboarding-test",
            "Status": "PENDING"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```


CloudTrail 中的 CloudWatch Synthetics 信息

CloudWatch Synthetics 支持在 CloudTrail 日志文件中将以下操作记录为事件：

- CreateCanary
- DeleteCanary
- DescribeCanaries
- DescribeCanariesLastRun
- DescribeRuntimeVersions
- GetCanary
- GetCanaryRuns
- ListTagsForResource
- StartCanary
- StopCanary
- TagResource
- UntagResource
- UpdateCanary

示例：CloudWatch Synthetics 日志文件条目

下面的示例显示了一个 CloudTrail Synthetics 日志条目，该条目说明了 `DescribeCanaries` 操作。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "DescribeCanaries",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "201ed5f3-15db-4f87-94a4-123456789",
    "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

下面的示例显示了一个 CloudTrail Synthetics 日志条目，该条目说明了 UpdateCanary 操作。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
               "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Schedule": {
            "Expression": "rate(1 minute)"
        },
        "name": "sample_canary_name",
        "Code": {
            "Handler": "myOwnScript.handler",
            "ZipFile": "SAMPLE_ZIP_FILE"
        }
    },
    "responseElements": null,
```

```
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

下面的示例显示了一个 CloudTrail Synthetics 日志条目，该条目说明了 GetCanaryRuns 操作。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "GetCanaryRuns",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Filter": "TIME_RANGE",
        "name": "sample_canary_name",
```

```
        "FilterValues": [
            "2020-04-08T23:00:00.000Z",
            "2020-04-08T23:10:00.000Z"
        ]
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CloudTrail 中的 CloudWatch RUM 信息

CloudWatch RUM 支持将以下操作记录为 CloudTrail 日志文件中的事件：

- BatchCreateRumMetricDefinitions
- BatchDeleteRumMetricDefinitions
- BatchGetRumMetricDefinitions
- CreateAppMonitor
- DeleteAppMonitor
- DeleteRumMetricsDestination
- GetAppMonitor
- GetAppMonitorData
- ListAppMonitors
- ListRumMetricsDestinations
- ListTagsForResource
- PutRumMetricsDestination
- TagResource
- UntagResource
- UpdateAppMonitor
- UpdateRumMetricDefinition

示例：CloudWatch RUM 日志文件条目

本节包含某些 CloudWatch RUM API 的 CloudTrail 条目示例。

以下示例是演示 CreateAppMonitor 操作的 CloudTrail 日志条目。


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:02:57Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "CreateAppMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "54.240.198.39",
    "userAgent": "aws-internal/3 aws-sdk-java/1.12.641
Linux/5.10.219-186.866.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/25.402-b08
java/1.8.0_402 vendor/Oracle_Corporation cfg/retry-mode/standard",
    "requestParameters": {
        "CustomEvents": {
            "Status": "ENABLED"
        },
        "CwLogEnabled": true,
        "Domain": "*.github.io",
        "AppMonitorConfiguration": {
```

```
            "SessionSampleRate": 1,
            "IncludedPages": [],
            "ExcludedPages": [],
            "Telemetries": [
                "performance",
                "errors",
                "http"
            ],
            "EnableXRay": false,
            "AllowCookies": true,
            "IdentityPoolId": "us-east-1:c81b9a1c-a5c9-4de5-8585-eb8df04e66f0"
        },
        "Tags": {
            "TestAppMonitor": ""
        },
        "Name": "TestAppMonitor"
    },
    "responseElements": {
        "Id": "65a8cc63-4ae8-4f2c-b5fc-4a54ef43af51"
    },
    "requestID": "cf7c30ad-25d3-4274-bab1-39c95a558007",
    "eventID": "2d43cc69-7f89-4f1a-95ae-0fc7e9b9fb3b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

以下示例是演示 PutRumMetricsDestination 操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
```

```
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:22:22Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "PutRumMetricsDestination",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.put-rum-metrics-destination",
    "requestParameters": {
        "Destination": "CloudWatch",
        "AppMonitorName": "TestAppMonitor"
    },
    "responseElements": null,
    "requestID": "9b03fcce-b3a2-44fc-b771-900e1702998a",
    "eventID": "6250f9b7-0505-4f96-9668-feb64f82de5b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

下面的示例显示了一个 CloudTrail 日志条目，该条目演示了 BatchCreateRumMetricsDefinitions 操作。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
```

```
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:23:11Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "BatchCreateRumMetricDefinitions",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.batch-create-rum-metric-definitions",
    "requestParameters": {
        "Destination": "CloudWatch",
        "MetricDefinitions": [
            {
                "Name": "NavigationToleratedTransaction",
                "Namespace": "AWS/RUM",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"]},
\"event_type\":[\"com.amazon.rum.performance_navigation_event\"],\"event_details\":
 {\"duration\": [{\"numeric\": [\"<=\",2000,\"<\",8000]}]}}"
            },
            {
                "Name": "HttpErrorCount",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName",
                    "metadata.countryCode": "CountryCode"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"],
 \"countryCode\":[\"US\"]},\"event_type\":[\"com.amazon.rum.http_event\"]}"
            }
        ],
        "AppMonitorName": "TestAppMonitor"
```

```
    },
    "responseElements": {
        "Errors": [],
        "MetricDefinitions": []
    },
    "requestID": "b14c5eda-f107-48e5-afae-1ac20d0962a8",
    "eventID": "001b55c6-1de1-48c0-a236-31096dffe249",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

CloudTrail 中的 CloudWatch RUM 数据面板事件

CloudTrail 可以捕获与 CloudWatch RUM 数据面板操作 PutRumEvents 相关的 API 活动。

数据事件也称为数据面板操作，可让您深入了解在资源上或资源内执行的资源操作。数据事件通常是高容量活动。

要在 CloudTrail 中启用 PutRumEvents 数据事件日志记录，需要在 CloudTrail 中启用数据面板 API 活动日志记录。有关更多信息，请参阅记录数据事件用于跟踪。

可以按资源类型筛选数据面板事件。由于在 CloudTrail 中使用数据事件会产生额外费用，因此按资源筛选可以让您更好地控制自己记录和支付费用的内容。

使用 CloudTrail 收集的信息，可以确定向 CloudWatch RUM PutRumEvents API 发出的特定请求、请求者的 IP 地址、请求者的身份以及请求的日期和时间。使用 CloudTrail 记录 PutRumEvents API 可帮助您实现 AWS 账户的运营和风险审计、治理与合规性。

以下示例显示一个 CloudTrail 日志条目，该条目演示 PutRumEvents 操作。

```
{
 "Records": [
     {
         "eventVersion": "1.09",
         "userIdentity": {
             "type": "AssumedRole",
             "principalId": "EXAMPLE_PRINCIPAL_ID",
             "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
             "accountId": "777777777777",
```

```
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "EXAMPLE_PRINCIPAL_ID",
                    "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                    "accountId": "777777777777",
                    "userName": "USERNAME_EXAMPLE"
                },
                "attributes": {
                    "creationDate": "2024-05-16T20:32:39Z",
                    "mfaAuthenticated": "false"
                }
            },
            "invokedBy": "AWS Internal"
        },
        "eventTime": "2024-05-16T20:32:42Z",
        "eventSource": "rum.amazonaws.com",
        "eventName": "PutRumEvents",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "AWS Internal",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "id": "73ddbd81-1234-5678-b246-123456789",
            "batchId": "123456-3dd0-4ada-b246-123456789",
            "appMonitorDetails": {
                "name": "APP-MONITOR-NAME",
                "id": "123456-3dd0-4ada-b246-123456789",
                "version": "1.0.0"
            },
            "userDetails": {
                "userId": "73ddbd81-1111-9999-b246-123456789",
                "sessionId": "a1b2c3456-15db-4f87-6789-123456789"
            },
            "rumEvents": [
                {
                    "id": "201f367a-15db-1234-94a4-123456789",
                    "timestamp": "May 16, 2024, 8:32:20 PM",
                    "type": "com.amazon.rum.dom_event",
                    "metadata": "{}",
                    "details": "{}"
                }
            ]
        },
```

```
        "responseElements": null,
        "requestID": "201ed5f3-15db-4f87-94a4-123456789",
        "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
        "readOnly": false,
        "resources": [
            {
                "accountId": "777777777777",
                "type": "AWS::RUM::AppMonitor",
                "ARN": "arn:aws:rum:us-east-1:777777777777:appmonitor/
APPMONITOR_NAME_EXAMPLE"
            }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "777777777777",
        "eventCategory": "Data"
    }
 ]
}
```

标记 Amazon CloudWatch 资源

标签是您或 AWS 分配给 AWS 资源的自定义属性标签。每个标签具有两个部分：

- 标签键（例如，`CostCenter`、`Environment` 或 `Project`）。标签键区分大小写。
- 一个称为标签值 的可选字段（例如，`111122223333` 或 `Production`）。省略标签值与使用空字符串效果相同。与标签键一样，标签值区分大小写。

标签可帮助您：

- 标识和整理您的 AWS 资源。许多 AWS 服务支持标记，因此，您可以将同一标签分配给来不同服务的资源，以指示这些资源是相关的。例如，您可以将您分配给 EC2 实例的同一标签分配给 CloudWatch 规则。

以下各节提供有关 CloudWatch 的标签的更多信息。

CloudWatch 中支持的资源

CloudWatch 中的以下资源支持标记：

- 警报 – 您可以使用 tag-resource AWS CLI 和 TagResource API 标记告警。您还可以通过 CloudWatch 控制台中的警报详细信息页面，来查看和管理警报标签。
- Canary – 您可以使用 CloudWatch 控制台标记 Canary。有关更多信息，请参阅创建金丝雀。
- Contributor Insights 规则 – 您可以在创建 Contributor Insights 规则时使用 put-insight-rule AWS CLI 命令和 PutInsightRule API 来标记它们。您可以使用 tag-resource AWS CLI 命令和 TagResource API 向现有规则添加标签。
- 指标流 – 您可以在创建指标流时使用 put-metric-stream AWS CLI 命令和 PutMetricStream API 来标记它们。您可以使用 tag-resource AWS CLI 命令和 TagResource API 向现有指标流添加标签。

有关添加和管理标签的信息，请参阅管理标签。

管理标签

标签由资源上的 `Key` 和 `Value` 属性构成。您可以使用 CloudWatch 控制台、AWS CLI 或 CloudWatch API 添加、编辑或删除这些属性的值。有关使用标签的信息，请参阅以下内容：

- Amazon CloudWatch API 引用中的 TagResource、UntagResource 和 ListTagsForResource
- Amazon CloudWatch CLI 引用中的 tag-resource、untag-resource 和 list-tags-for-resource
- Resource Groups 用户指南中的使用标签编辑器

标签命名和使用约定

以下基本命名和使用约定适用于将标签与 CloudWatch 资源一起使用的情况：

- 每个资源最多可以有 50 个标签。
- 对于每个资源，每个标签键都必须是唯一的，每个标签键只能有一个值。
- 最大标签键长度为 128 个 Unicode 字符 (采用 UTF-8 格式)。
- 最大标签值长度为 256 个 Unicode 字符 (采用 UTF-8 格式)。
- 允许使用的字符包括可用 UTF-8 格式表示的字母、数字和空格，以及以下字符：. : + = @ _ / - (连字符) 。
- 标签键和值区分大小写。最佳实践是，决定利用标签的策略并在所有资源类型中一致地实施该策略。例如，决定是否使用 `Costcenter`、`costcenter` 或 `CostCenter`，以及是否对所有标签使用相同的约定。避免将类似的标签用于不一致的案例处理。
- 对标签禁止使用 `aws:` 前缀，因为它是为使用 AWS 而保留的。您无法编辑或删除带此前缀的标签键或值。具有此前缀的标签不计入每个资源的标签数限制。

Grafana 集成

您可以使用 Grafana 版本 6.5.0 及更高版本，通过 CloudWatch 控制台在上下文中前进，并使用通配符查询指标的动态列表。这可以帮助您监控 AWS 资源（例如 Amazon Elastic Compute Cloud 实例或容器）的指标。当新实例作为 Auto Scaling 事件的一部分创建时，它们将自动显示在图形中。您无需跟踪新实例 ID。预构建的控制面板可帮助简化监控 Amazon EC2、Amazon Elastic Block Store 和 AWS Lambda 资源的入门体验。

您可以使用 Grafana 版本 7.0 及更高版本对 CloudWatch Logs 中的日志组执行 CloudWatch Logs Insights 查询。您可以在条形图、折线图和堆叠图形中以及以表格格式显示查询结果。有关 CloudWatch Logs Insights 的更多信息，请参阅使用 CloudWatch Logs Insights 分析日志数据。

有关如何入门的更多信息，请参阅 Grafana Labs 文档中的在 Grafana 中使用 AWS CloudWatch。

CloudWatch 服务配额

CloudWatch 对指标、告警、API 请求和告警电子邮件通知有以下配额。

> **Note**
>
> 对于一些包含 CloudWatch 的 AWS 服务，您可以使用 CloudWatch 使用情况指标在 CloudWatch 图表和控制面板上直观呈现当前服务使用情况。您可以使用 CloudWatch 指标数学函数在图表上显示这些资源的服务配额。还可以配置警报，以在用量接近服务配额时向您发出警报。有关更多信息，请参阅 可视化 Service Quotas 并设置告警。

资源	默认配额
告警操作	每个警报 5 个。无法更改此配额。
告警评估期	通过将告警周期乘以使用的评估周期数计算的最大值为 1 天（86,400 秒）。无法更改此配额。
告警	每月为每个客户免费提供 10 个。额外的告警会产生费用。 每个账户的告警总数没有限制。 基于指标数学表达式的警报可以拥有最多 10 个指标。 每个区域 200 个 Metrics Insights 警报。您可以请求提高限额。
异常检测模型	每账户每区域 500 个。
API 请求数量	每月为每个客户免费提供 1,000,000 个。
金丝雀	每账户每区域 200 个。 您可以请求提高限额。
Contributor Insights API 请求	以下 API 具有每个区域每秒 20 个事务（TPS）的配额。 • DescribeInsightRules

资源	默认配额
	此配额无法更改。 • GetInsightRuleReport 您可以请求提高限额。 以下 API 具有每个区域 5 TPS 的配额。无法更改此配额。 • DeleteInsightRules • PutInsightRule 以下 API 具有每个区域 1 TPS 的配额。无法更改此配额。 • DisableInsightRules • EnableInsightRules
Contributor Insights 规则	每区域每账户 100 条规则。 您可以请求提高限额。
自定义指标	没有配额。
控制面板	每个控制面板最多 500 个小部件。每个控制面板小部件最多 500 个指标。每个控制面板中的所有小部件最多 2500 个指标。 这些配额包括检索的用于指标数学函数的所有指标，即使这些指标未显示在图表上也是如此。 这些配额不能更改。
DescribeAlarms	每个区域 9 个事务每秒 (TPS)。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。

资源	默认配额
DeleteAlarms 请求 DescribeAlarmHistory 请求 DisableAlarmActions 请求 EnableAlarmActions 请求 SetAlarmState 请求	对于其中每个操作，每个区域 3 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 这些配额不能更改。
DescribeAlarmsForMetric 请求	每个区域 9 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 无法更改这些配额。
DeleteDashboards 请求 GetDashboard 请求 ListDashboards 请求 PutDashboard 请求	对于其中每个操作，每个区域 10 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 这些配额不能更改。
PutAnomalyDetector DescribeAnomalyDetectors	每个区域 10 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。
DeleteAnomalyDetector	每个区域 5 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。
尺寸	30/指标。无法更改此配额。

资源	默认配额
GetMetricData	对于包含 Metrics Insights 查询的操作，每个区域 10 TPS。对于不包含 Metrics Insights 查询的操作，配额为每个区域 50 TPS。这是您在不受限制的情况下，每秒可发出的最大操作请求数。您可以请求提高限额。 对于包含 Metrics Insights 查询的 GetMetricData 操作，最近 3 小时的配额为每秒 4,300,000 个数据点 (DPS)。这是根据查询扫描的数据点总数计算得出的（其中可包括最多 10,000 个指标。） 如果在 API 中使用的 StartTime 小于或等于从当前时间开始 3 小时，则为 180,000 每秒数据点 (DPS)。如果 StartTime 大于从当前时间开始 3 小时，则为 396,000 DPS。这是您可以使用一个或多个 API 调用 (而不被限制) 每秒请求的最大数据点数。无法更改此配额。 DPS 是根据估计数据点而非实际数据点计算的。使用请求的时间范围、时间段和保留期计算数据点估计值。这意味着，如果请求指标中的实际数据点稀疏或为空，那么，当估计数据点超过配额时，仍会发生限制。DPS 配额是按每个区域计算的。
GetMetricData	信号 GetMetricData 调用可包含： • 多达 500 个 MetricDataQuery 结构。 • 多达 100 个 SERVICE_QUOTA() 函数。 • 多达 100 个 SEARCH() 函数。 • 多达 5 个 LAMBDA() 函数。 无法更改这些限额。
GetMetricStatistics	每个区域 400 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。

资源	默认配额
GetMetricWidgetImage	每幅图像最多 500 个指标。无法更改此配额。 每个区域 20 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。
ListMetrics	每个区域 25 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。
指标数据值	指标数据点的值必须在 -2^360 到 2^360 的范围内。不支持特殊值（例如，NaN、+无穷大、-无穷大）。无法更改此配额。
MetricDatum 项目	每个 PutMetricData 请求 1000 个。一个 MetricDatum 对象可包含一个值或代表多个值的一个 StatisticSet 对象。无法更改此配额。
指标	每月为每个客户免费提供 10 个。
Metrics Insights 查询	单个查询可处理的指标不超过 10,000 个。这意味着，如果 SELECT、FROM 和 WHERE 子句匹配 10,000 个以上指标，查询只会处理找到的前 10,000 个指标。 单个查询可返回的时间序列不超过 500 个。 您只能查询最近三个小时的数据
Metrics Insights 查询生成器请求	最多五个并发的自然语言生成请求。

资源	默认配额
Observability Access Manager（OAM）API 请求速率。	PutSinkPolicy 的每个区域为 1 TPS。 每个其他 CloudWatch OAM API 的每个区域为 10 TPS。 这些限额显示了在不节流的情况下，您每秒可发出的操作请求的最大数目。 无法更改这些限额。
OAM 源账户关联	每个源账户最多可以关联五个监控账户。 无法更改此配额。
OAM 接收器	每账户每区域 1 个接收器 无法更改此配额。
PutCompositeAlarm 请求	每个区域 3 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。
PutMetricAlarm 请求	每个区域 3 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 您可以请求提高限额。
PutMetricData 请求	对于 HTTP POST 请求为 1 MB。PutMetricData 每秒可处理 500 个事务 (TPS)，这是您每秒可以发出而不会受到限制的操作请求的最大数量。`PutMetricData` 可以处理每个请求 1,000 个指标。 您可以请求提高限额。
Amazon SNS 电子邮件通知	每月为每个客户免费提供 1,000 个。
Synthetics Group	每个账户 20。 无法更改此配额。

资源	默认配额
TagResource	每个区域 20 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 无法更改此配额。
UntagResource	每个区域 20 TPS。在不受限制的情况下，每秒可发出的操作请求的最大数目。 无法更改此配额。

文档历史记录

下表列出了从 2018 年 6 月开始的每个版本的 Amazon CloudWatch 用户指南中的重要更改。要获得本文档的更新通知，您可以订阅 RSS 源。

变更	说明	日期
网络监测仪更新了控制面板和延迟改善建议	Amazon CloudWatch 网络监测仪已推出更新的控制台体验，包括用于可视化配置更改的新功能，这些功能有助您减少应用程序的延迟。	2024 年 8 月 12 日
Application Signals 对 .NET 应用程序的支持预览版	CloudWatch Application Signals 增加了对 .NET 应用程序的支持，可在支持的平台上使用适用于 .NET 的 ADOT 仪表化工具。此特征提供预览版。	2024 年 8 月 5 日
已添加新的 CloudWatch Application Insights 服务相关角色权限	允许 CloudWatch Application Insights 启用和禁用 AWS CloudFormation 堆栈上的终止保护。	2024 年 7 月 25 日
CloudWatch Metrics Insights 对自然语言查询生成的支持已正式发布。	CloudWatch Metrics Insights 支持使用自然语言查询来生成与更新查询。有关更多信息，请参阅使用自然语言生成与更新 CloudWatch Metric Insights 查询。	2024 年 6 月 10 日
CloudWatch Application Signals 已正式发布	CloudWatch Application Signals 现已正式发布。通过使用 Application Signals 来检测 AWS 上的应用程序，您可以监控应用程序的当前运行状况、	2024 年 6 月 10 日

	创建服务级别目标（SLO），并根据业务目标跟踪应用程序的长期性能。有关更多信息，请参阅 Application Signals。	
新的 CloudWatchApplicationSignalsReadOnlyAccess IAM 策略	CloudWatch 添加了 CloudWatchApplicationSignalsReadOnlyAccess 策略，以在 CloudWatch 控制台中为 Application Signals 添加可用的只读操作和资源。	2024 年 6 月 7 日
新的 CloudWatchApplicationSignalsFullAccess 策略	CloudWatch 添加了 CloudWatchApplicationSignalsFullAccess 策略，以在 CloudWatch 控制台中为 Application Signals 添加可用的操作和资源。	2024 年 6 月 7 日
CloudTrail 现在可以捕获与 CloudWatch 数据面板操作相关的 API 活动。	CloudTrail 现在可在 CloudTrail 中记录 GetMetricData 和 GetMetricWidgetImage API 活动的事件。	2024 年 6 月 6 日
CloudWatch Application Signals 服务地图支持 Canary、RUM 客户端和 AWS 服务依赖项分组。	Application Signals 预览版在服务地图中为同类型的加那利群岛、RUM 客户端和 AWS 服务依赖项添加了默认分组。此更改减少了服务地图默认视图中的图标数量，使其更易于查看和导航。	2024 年 5 月 21 日

CloudWatchReadOnlyAccess 策略已更新	CloudWatch 更改了 CloudWatchReadOnlyAccess 中权限的范围。该策略的范围添加了 `application-signals:BatchGet*` 、`application-signals:Get*` 和 `application-signals:List*` 操作，以便用户可以使用 CloudWatch Application Signals 来查看、调查和诊断其服务运行状况的问题。CloudWatch 还添加了一个 `iam:GetRole` 操作，以便用户可以检查 Application Signals 是否已设置。	2024 年 5 月 17 日
CloudWatchFullAccessV2 策略已更新	CloudWatch 已更改 CloudWatchFullAccessV2 中的权限范围。该策略的范围添加 `application-signals:*` ，以便用户可以使用 CloudWatch Application Signals 来查看、调查和诊断其服务运行状况的问题。	2024 年 5 月 17 日
Lambda Insights 支持 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）	CloudWatch Lambda Insights 增加了对 AWS GovCloud（美国东部）和 AWS GovCloud（美国西部）区域的支持。	2024 年 4 月 29 日
CloudWatch 跨账户可观测性支持资源筛选器	现在，创建账户之间的链接时，您可以创建筛选器来指定源账户与监控账户共享哪些指标命名空间和日志组。	2024 年 4 月 26 日

CloudWatch Application Signals 更新	Application Signals 预览版增加了三项功能。Application Signals 现在支持 Python 应用程序。它为 Amazon EKS 架构上的应用程序提供更简单的启用流程。它还包括新配置，您可以使用这些配置来管理所收集指标的基数。	2024 年 4 月 26 日
针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 可以收集 AWS Elastic Fabric Adapter（EFA）指标	现在，您可以使用针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 从 Amazon EKS 集群收集 AWS Elastic Fabric Adapter（EFA）指标。	2024 年 4 月 23 日
已更新 IAM 策略	CloudWatch 更新了 CloudWatchApplicationSignalsServiceRolePolicy 策略。此策略中 `logs:StartQuery` 和 `logs:GetQueryResults` 权限的范围已更改，添加 `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` 和 `"arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"` ，以在更多架构上启用 Application Signals。此策略附加到 AWSServiceRoleForCloudWatchApplicationSignals 服务相关角色。	2024 年 4 月 18 日

网络监测仪为经身份验证的 AWS 客户提供全球互联网天气地图	Amazon CloudWatch 网络监测仪现在显示全球互联网天气地图，所有经过身份验证的 AWS 客户都可以在控制台中使用该地图。要查看地图，请在 Amazon CloudWatch 控制台中导航到网络监测仪。	2024 年 4 月 16 日
针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 可以收集 AWS Neuron 指标	现在，您可以使用针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 从 Amazon EKS 集群收集 AWS Neuron 指标。	2024 年 4 月 16 日
CloudWatch Application Signals 添加了服务概述选项卡和更多指标来帮助诊断	新的服务概述选项卡显示您的服务概述，包括操作数量、依赖项、Synthetics 和客户端页面。该选项卡显示整个服务、顶部操作和依赖项的关键指标。现在，您还可以查看与故障、错误和延迟问题等问题相关的 X-Ray 跟踪。	2024 年 4 月 16 日
针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 添加了对 Windows 的支持	现在，您可以使用针对 Amazon EKS 增强了可观测性的 CloudWatch Container Insights 从 Amazon EKS 集群上的 Windows Worker 节点收集指标。	2024 年 4 月 10 日

CloudWatchApplicationSignalsServiceRolePolicy IAM 策略已更新	CloudWatch 更改了 CloudWatchApplicationSignalsServiceRolePolicy 中权限的范围。`cloudwatch:GetMetricData` 权限的范围已更改为 `*`，以使 Application Signals 可以从关联账户中的源检索指标。	2024 年 4 月 8 日
Amazon CloudWatch 网络监测仪现在支持跨账户可观测性	现在，您可以借助网络监测仪跨账户可观测性来监测单个 AWS 区域 中跨多个 AWS 账户的应用程序。	2024 年 3 月 29 日
CloudWatchAgentServerPolicy 和 CloudWatchAgentAdminPolicy 策略已更新	CloudWatch 为 `CloudWatchAgentServerPolicy` 和 `CloudWatchAgentAdminPolicy` 策略添加了权限，允许 CloudWatch 代理发布 X-Ray 跟踪和修改日志组保留期。在这两个策略中，都添加了 `xray:PutTraceSegments`、`xray:PutTelemetryRecords`、`xray:GetSamplingRules`、`xray:GetSamplingTargets`、`xray:GetSamplingStatisticSummaries` 和 `logs:PutRetentionPolicy` 权限	2024 年 2 月 12 日

CloudWatch 网络监测仪的新服务关联角色和 IAM 策略	CloudWatch 添加了一个新的服务相关角色，名为 AWSServiceRoleForNetworkMonitor。CloudWatch 添加该服务相关角色后，您可以创建监测仪来获取源子网和目标 IP 地址之间的网络指标。此角色附加了一个新 CloudWatchNetworkMonitorServiceRolePolicy IAM 策略，该策略授予 CloudWatch 代表您获取网络指标的权限。	2023 年 12 月 22 日
CloudWatch 发布 Amazon CloudWatch 网络监测仪	CloudWatch 发布了一项新功能，即 Amazon CloudWatch 网络监测仪。这是一项新的活动网络监控服务，可识别 AWS 网络中以及您自己的公司网络中是否存在网络问题。	2023 年 12 月 22 日
CloudWatchReadOnlyAccess 策略已更新	CloudWatch 为 CloudWatch Synthetics、X-Ray 和 CloudWatch RUM 添加了现有的只读权限，并为 CloudWatch Application Signals 添加了新的只读权限 CloudWatchReadOnlyAccess，以便使用此策略的用户可以根据 CloudWatch Application Signals 的报告来分类和诊断服务运行状况问题。添加了 `cloudwatch:GenerateQuery` 权限，以便拥有此策略的用户可以根据自然语言提示来生成 CloudWatch Metrics Insights 查询字符串。	2023 年 12 月 5 日

CloudWatchFullAccessV2 策略已更新	对于 CloudWatch Synthetics、X-Ray 和 CloudWatch RUM，CloudWatch 向 CloudWatch FullAccessV2 添加了现有权限，并为 CloudWatch Application Signals 添加了新权限，以便使用此策略的用户可以完全管理 Application Signals，从而进行分类并诊断服务运行状况的问题。	2023 年 12 月 5 日
新的服务相关角色和新的 IAM 策略	CloudWatch 添加了一个新的服务相关角色，名为 AWSServiceRoleForCloudWatchApplicationSignals。CloudWatch 添加了这个新的服务相关角色，以允许 CloudWatch Application Signals 收集 CloudWatch Logs 数据、X-Ray 跟踪数据、CloudWatch 指标数据，以及您为 CloudWatch Application Signals 启用的应用程序中的标记数据。此角色附加了一个新 CloudWatchApplicationSignalsServiceRolePolicy IAM 策略，该策略授予 CloudWatch Application Signals 从其他相关 AWS 服务收集监控和标记数据的权限。	2023 年 11 月 30 日

CloudWatch 推出 Application Signals 预览版	CloudWatch Application Signals 目前为预览版。通过使用 Application Signals 来检测 AWS 上的应用程序，您可以监控应用程序的当前运行状况、创建服务级别目标（SLO），并根据业务目标跟踪应用程序的长期性能。有关更多信息，请参阅 Application Signals。	2023 年 11 月 30 日
CloudWatch 添加对查询其他数据来源的支持	您可以使用 CloudWatch 对源自其他数据来源的指标进行查询、可视化并为其创建警报。有关更多信息，请参阅查询来自其他数据来源的指标。	2023 年 11 月 26 日
CloudWatch Metrics Insights 支持自然语言查询生成	CloudWatch Metrics Insights 支持使用自然语言查询来生成与更新查询。有关更多信息，请参阅使用自然语言生成与更新 CloudWatch Metric Insights 查询。	2023 年 11 月 26 日
CloudWatch 发布针对 Amazon EKS 增强了可观测性的 Container Insights	CloudWatch 发布了新版本的 Container Insights。此版本支持针对 Amazon EKS 集群增强的可观测性，并且可以从运行 Amazon EKS 的集群收集更详细的指标。安装后，它会自动为您的 Amazon EKS 集群收集详细的基础设施遥测数据和容器日志。然后，您可以使用精选的、可立即使用的控制面板，深入研究应用程序和基础设施的遥测数据。	2023 年 11 月 6 日

CloudWatch 指标流新增合作伙伴快速设置	CloudWatch 指标流现在提供合作伙伴快速设置选项，您可以使用该选项快速设置要传输到某些第三方提供商的指标流。	2023 年 10 月 17 日
CloudWatch 发布警报建议	CloudWatch Synthetics 现在会为其他 AWS 服务的指标提供警报建议。这些建议可以帮助您确定应为其设置警报的指标，以遵循监控这些服务的最佳实践。	2023 年 10 月 16 日
CloudWatch Synetics 发布运行时系统 syn-nodejs-puppeteer-6.0	CloudWatch Synthetics 发布了运行时系统 `syn-nodejs-puppeteer-6.0` 。	2023 年 9 月 26 日
Amazon CloudWatch Application Insights 增加跨账户应用程序支持	现在，您可以跨越账户边界共享 CloudWatch Application Insights 应用程序。	2023 年 9 月 26 日
新的服务相关角色和新的 IAM 策略	CloudWatch 添加了一个新的服务相关角色，名为 AWSServiceRoleForCloudWatchMetrics_DbPerfInsights。CloudWatch 添加了这个服务相关角色，允许 CloudWatch 获取用于警报、异常检测和快照的性能详情指标。此角色附加了新的 AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy IAM 策略，该策略授予 CloudWatch 代表您获取性能详情指标的权限。	2023 年 9 月 20 日

新增指标数学函数	CloudWatch 增加了新的指标数学函数 DB_PERF_INSIGHTS ，您可以使用该函数从 AWS 数据库服务中获取用于警报、异常检测和快照的性能详情指标。	2023 年 9 月 20 日
CloudWatchReadOnlyAccess 策略已更新	CloudWatch 向 CloudWatchReadOnlyAccess 添加了 application-autoscaling:DescribeScalingPolicies 权限，因此拥有此策略的用户可以访问有关 Application Auto Scaling 策略的信息。	2023 年 9 月 14 日
CloudWatch 代理增加了对 AL2023 的支持	CloudWatch 代理支持 AL2023。	2023 年 8 月 8 日
新的托管式 IAM 策略CloudwatchFullAccess	CloudWatch 添加了一项新策略 CloudWatchFullAccessV2。此策略授予对 CloudWatch 操作和资源的完全访问权限，同时可以更好地限定授予其他服务（例如 Amazon SNS 和 Amazon EC2 Auto Scaling）的权限。	2023 年 8 月 1 日
更新了 Amazon CloudWatch 网络监测仪的服务关联角色 — 更新到现有政策	为网络监测仪的服务关联角色添加了新权限 elasticloadbalancing:DescribeLoadBalancers 和 ec2:DescribeNetworkInterfaces ，以支持监测特定网络负载均衡器资源的流量。	2023 年 7 月 25 日

在 Amazon CloudWatch 网络监测仪中添加了对网络负载均衡器资源的支持	添加对使用特定网络负载均衡器资源在网络监测仪中创建监测仪的支持，以便为您的应用程序提供更精细的可观测性级别。	2023 年 7 月 25 日
控制面板变量功能	CloudWatch 发布了控制面板变量，您可以借助此功能创建灵活的控制面板，从而可以根据您在控制面板中设置输入字段的方式，来快速显示不同的内容。例如，您可以创建一个可在不同 Lambda 函数或 Amazon EC2 实例 ID 之间快速切换的控制面板，也可以创建一个可以切换到不同 AWS 区域的控制面板。有关更多信息，请参阅使用控制面板变量创建灵活的控制面板。	2023 年 6 月 28 日
网络监测仪现在支持自定义运行状况事件的阈值	网络监测仪新增了功能，允许自定义全局性能分数或可用性分数将触发运行状况事件的阈值。有关更多信息，请参阅 在 Amazon CloudWatch 网络监测仪中跟踪实时性能和可用性。	2023 年 6 月 26 日
网络监测仪现在支持所有商业区域	网络监测仪新增支持七个新的 AWS 区域，现已支持所有商业区域。	2023 年 6 月 19 日
新 Lambda Insights 扩展程序版本	CloudWatch 为 x86-64 平台和 ARM64 平台增加了 1.0.229.0 版本的 Lambda Insights 扩展程序。有关更多信息，请参阅 Lambda Insights 扩展程序的可用版本。	2023 年 6 月 12 日

CloudWatchReadOnlyAccess 策略已更新	CloudWatch 添加了对 CloudWatchReadOnlyAccess 的权限。增加了 `logs:StartLiveTail` 和 `logs:StopLiveTail` 权限，以便使用此策略的用户可以使用控制台启动和停止 CloudWatch Logs Live Tail 会话。有关更多信息，请参阅 使用 Live Tail 近乎实时地查看日志。	2023 年 6 月 6 日
CloudWatch RUM 增加了对自定义指标的支持	您可以使用 CloudWatch RUM 应用程序监控来创建自定义指标，并将其发送到 CloudWatch 和 CloudWatch Evidently。此功能包括对 AmazonCloudWatchRUMServiceRolePolicy 托管 IAM 策略的更新。在该策略中，条件键已更改，以便 CloudWatch RUM 可以将自定义指标发送到自定义指标命名空间。	2023 年 2 月 9 日
CloudWatch 的新增和更新的托管式策略	为支持 CloudWatch 跨账户可观测性，已更新 `CloudWatchFullAccess` 和 `CloudWatchReadOnlyAccess` 策略，并添加了以下新的托管策略：`CloudWatchCrossAccountSharingConfiguration` 、`OAMFullAccess` 和 `OAMReadOnlyAccess` 。有关更多信息，请参阅 CloudWatch 对 AWS 托管式策略做出的更新。	2023 年 2 月 7 日

CloudWatch Appilivery Insights 服务相关角色策略更新 – 对现有策略的更新。	CloudWatch Application Insights 更新了现有 AWS 服务相关角色策略。	2022 年 12 月 19 日
Amazon CloudWatch Application Insights 对 Container Insights 控制台中的容器化应用程序和微服务的支持。	Container Insights 控制面板上可以显示 CloudWatch Application Insights 检测到的 Amazon ECS 和 Amazon EKS 的问题。	2021 年 11 月 17 日
适用于 SAP HANA 数据库的 Amazon CloudWatch Application Insights 监控。	您可以使用 Application Insights 监控 SAP HANA 数据库。	2021 年 11 月 15 日
Amazon CloudWatch Application Insights 对监控账户中的所有资源的支持。	您可以载入和监控账户中的所有资源。	2021 年 9 月 15 日
Amazon CloudWatch Application Insights 对 Amazon FSx 的支持。	您可以监控从 Amazon FSx 检索的指标。	2021 年 8 月 31 日
不再支持软件开发工具包指标。	不再支持 CloudWatch 软件开发工具包指标。	2021 年 8 月 25 日
Amazon CloudWatch Application Insights 对设置容器监控的支持。	您可以使用 Amazon CloudWatch Application Insights 最佳实践监控容器。	2021 年 5 月 18 日
已正式发布指标流	您可以使用指标流，持续地将 CloudWatch 指标流式传输到您所选的目标位置。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的指标流。	2021 年 3 月 31 日

适用于 Amazon RDS 和 Amazon EC2 上的 Oracle 数据库的 Amazon CloudWatch Application Insights 监控。	您可以使用 Amazon CloudWatch Application Insights 监控从 Oracle 检索到的指标和日志。	2021 年 1 月 16 日
Lambda Insights 已正式发布	CloudWatch Lambda Insights 是针对在 AWS Lambda 上运行的无服务器应用程序的监控和故障排除解决方案。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 Lambda Insights。	2020 年 12 月 3 日
适用于 Prometheus JMX Exporter 指标的 Amazon CloudWatch Application Insights 监控。	您可用使用 Amazon CloudWatch Application Insights 监控 从 Prometheus JMX Exporter 检索到的指标。	2020 年 11 月 20 日
CloudWatch Synthetics 发布新的运行时版本	CloudWatch Synthetics 已发布新的运行时版本。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的 Canary 运行时版本。	2020 年 9 月 11 日
适用于 Amazon RDS 和 Amazon EC2 上的 Postgre SQL 的 Amazon CloudWatch Application Insights 监控。	您可以监控运行在 Amazon RDS 或 Amazon EC2 上的使用 PostgreSQL 构建的应用程序。	2020 年 9 月 11 日
CloudWatch 支持控制面板共享	现在，您可以与企业和 AWS 账户之外的人员共享 CloudWatch 控制面板。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的共享 CloudWatch 控制面板。	2020 年 9 月 10 日

通过 CloudWatch Application Insights 在后端使用 SQL Server 为 .NET 应用程序设置监控	您可用使用文档教程帮助您通过 CloudWatch Application Insights 在后端使用 SQL Server 为 .NET 应用程序设置监控。	2020 年 8 月 19 日
AWS CloudFormation 对 Amazon CloudWatch Application Insights 应用程序的支持。	您可以直接从 AWS CloudFormation 模板中将 CloudWatch Application Insights 监控（包括关键指标和遥测）添加到应用程序、数据库和 Web 服务器。	2020 年 7 月 30 日
适用于 MySQL 数据库集群可用的 Aurora 的 Amazon CloudWatch Application Insights 监控。	您可用使用 Amazon CloudWatch Application Insights 监控适用于 MySQL 数据库集群 (RDS Aurora) 的 Aurora。	2020 年 7 月 2 日
CloudWatch Contributor Insights 正式发布	CloudWatch Contributor Insights 现已正式发布。它使您能够分析日志数据并创建显示贡献者数据的时间序列。您可以查看有关前 N 个贡献者、独特贡献者总数及其使用情况的指标。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的 使用 Contributor Insights 分析高基数数据。	2020 年 4 月 2 日
CloudWatch Synthetics 公开预览版	CloudWatch Synthetics 现在处于公开预览版。它使您能够创建 Canary 来监控终端节点和 API。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 Canary。	2019 年 11 月 25 日

CloudWatch Contributor Insights 公开预览版	CloudWatch Contributor Insights 现在处于公开预览版。它使您能够分析日志数据并创建显示贡献者数据的时间序列。您可以查看有关前 N 个贡献者、独特贡献者总数及其使用情况的指标。有关更多信息，请参阅《Amazon CloudWatch 用户指南》中的 使用 Contributor Insights 分析高基数数据。	2019 年 11 月 25 日
CloudWatch 启动 ServiceLens 功能	ServiceLens 使您能够将跟踪、指标、日志和警报集成到一个位置，从而提高服务和应用程序的可观察性。ServiceLens 将 CloudWatch 与 AWS X-Ray 集成，以提供应用程序的端到端视图。	2019 年 11 月 21 日
使用 CloudWatch 主动管理您的 AWS 服务限额	使用 CloudWatch 主动管理您的 AWS 服务配额。CloudWatch 使用情况指标可帮助您了解您账户的资源使用情况和 API 操作。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的 Service Quotas 集成和使用情况指标。	2019 年 11 月 19 日
当告警状态更改时，CloudWatch 会发送事件	现在，当任何 CloudWatch 告警更改状态时，CloudWatch 会向 Amazon EventBridge 发送事件。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的告警事件和 EventBridge。	2019 年 10 月 8 日

Container Insights	CloudWatch Container Insights 现已正式发布。它让您能够从容器化应用程序和微服务中收集、聚合和汇总指标与日志。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 Container Insights。	2019 年 8 月 30 日
Amazon EKS 和 Kubernetes 上的 Container Insights 预览指标更新	Amazon EKS 和 Kubernetes 上的 Container Insights 公开预览版已更新。InstanceId 现在作为一个维度包含到集群 EC2 实例中。这允许在这些指标上创建的警报触发以下 EC2 操作：停止、终止、重新启动或恢复。此外，Kubernetes 命名空间现在报告了 pod 和服务指标，以简化命名空间对指标的监控和警报。	2019 年 8 月 19 日
针对 AWS Systems Manager OpsCenter 集成的更新	关于 CloudWatch Application Insights 如何与 Systems Manager OpsCenter 集成的更新。	2019 年 8 月 7 日
CloudWatch 使用情况指标	CloudWatch 使用情况指标可帮助您跟踪 CloudWatch 资源的使用情况，并保持在服务限制范围内。有关更多信息，请参阅 https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html。	2019 年 8 月 6 日

CloudWatch Container Insights 公开预览版	CloudWatch Container Insights 现在处于公开预览版。它让您能够从容器化应用程序和微服务中收集、聚合和汇总指标与日志。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 Container Insights。	2019 年 7 月 9 日
CloudWatch 异常检测公开预览版	CloudWatch 异常检测现在处于公开预览版。CloudWatch 将机器学习算法应用于指标的过去数据，以创建指标的预期值模型。您可以使用此模型进行可视化和设置警报。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 CloudWatch 异常检测。	2019 年 7 月 9 日
适用于 .NET 和 SQL Server 的 CloudWatch Application Insights	适用于 .NET 和 SQL Server 的 CloudWatch Application Insights 简化了观察 .NET 和 SQL Server 应用程序的过程。它可以帮助您为应用程序资源设置最佳的监视器以持续分析数据，以便找出应用程序出现问题的迹象。	2019 年 6 月 21 日
已重新组织 CloudWatch 代理部分	已重新编写 CloudWatch 代理文档以提高清晰性，尤其是对于使用命令行安装和配置代理的客户。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 CloudWatch 代理从 Amazon EC2 实例和本地服务器中收集指标和日志。	2019 年 3 月 28 日

SEARCH 函数已添加到指标数学表达式	现在，您可以在指标数学表达式中使用 SEARCH 函数。这使您能够创建在创建与搜索查询匹配的新资源时自动更新的控制面板。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的在图表中使用搜索表达式。	2019 年 3 月 21 日
企业支持的 AWS SDK 指标	软件开发工具包指标可帮助您评估 AWS 服务的运行状况和诊断因达到账户用量限制或服务中断而引起的延迟。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 AWS SDK 指标监控应用程序。	2018 年 12 月 11 日
针对数学表达式的告警	CloudWatch 支持根据指标数学表达式创建告警。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的针对数学表达式的告警。	2018 年 11 月 20 日
新的 CloudWatch 控制台主页	Amazon 在 CloudWatch 控制台中创建了一个新的主页，此主页会自动显示您所使用的所有 AWS 服务的关键指标和告警。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的 Amazon CloudWatch 入门。	2018 年 11 月 19 日

CloudWatch 代理的 AWS CloudFormation 模板	Amazon 已上传 AWS CloudFormation 模板，您可以使用这些模板安装和更新 CloudWatch 代理。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 AWS CloudFormation 在新实例上安装 CloudWatch 代理。	2018 年 11 月 9 日
CloudWatch 代理的增强功能	CloudWatch 代理已更新为使用 StatsD 和 collectd 协议。它还改进了跨账户支持。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 StatsD 检索自定义指标、使用 collectd 检索自定义指标和将指标和日志发送到不同 AWS 账户。	2018 年 9 月 28 日
对 Amazon VPC 终端节点的支持	您现在可以在 VPC 和 CloudWatch 之间建立私有连接。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的将 CloudWatch 与接口 VPC 终端节点结合使用。	2018 年 6 月 28 日

下表介绍了 2018 年 6 月之前对 Amazon CloudWatch 用户指南的重要更改。

更改	描述	发行日期
指标数学	现在，您可以对 CloudWatch 指标执行数学表达式，从而生成可添加到控制面板上的图表中的新时间序列。有关更多信息，请参阅 使用指标数学。	2018 年 4 月 4 日

更改	描述	发行日期
“M out of N”警报	现在，您可以将警报配置为根据任何警报评估间隔的 N 个数据点中的 M 个数据点触发。有关更多信息，请参阅 评估告警。	2017 年 12 月 8 日
CloudWatch 代理	已发布新的统一 CloudWatch 代理。您可以使用统一多平台代理从 Amazon EC2 实例和本地服务器中收集自定义系统指标和日志文件。新代理支持 Windows 和 Linux 并允许自定义收集的指标，包括子资源指标，例如，每个 CPU 的内核数。有关更多信息，请参阅 使用 CloudWatch 代理收集指标、日志和跟踪信息。	2017 年 9 月 7 日
NAT 网关指标	已为 Amazon VPC NAT 网关添加指标。	2017 年 9 月 7 日
高精度指标	现在，您可以选择将自定义指标设置为高精度指标，其粒度可以低至一秒。有关更多信息，请参阅 高精度指标。	2017 年 26 月 7 日
控制面板 API	现在您可以使用 API 和 AWS CLI 创建、修改和删除控制面板。有关更多信息，请参阅 创建 CloudWatch 控制面板。	2017 年 6 月 7 日
AWS Direct Connect 指标	为 AWS Direct Connect 添加了指标。	2017 年 6 月 29 日
Amazon VPC VPN 指标	已为 Amazon VPC VPN 添加指标。	2017 年 5 月 15 日
AppStream 2.0 指标	已为 AppStream 2.0 添加指标。	2017 年 3 月 8 日
CloudWatch 控制台颜色选取器	您现在可以在控制面板小部件上选择每个指标的颜色。有关更多信息，请参阅 在 CloudWatch 控制面板上编辑图表 。	2017 年 2 月 27 日

更改	描述	发行日期
控制面板上的警报	现在可将警报添加到控制面板。有关更多信息，请参阅 在 CloudWatch 控制面板上添加或删除警报小组件 。	2017 年 2 月 15 日
已为 Amazon Polly 添加指标	已为 Amazon Polly 添加指标。	2016 年 12 月 1 日
增加了适用于 Apache Flink 的亚马逊托管服务的指标	增加了适用于 Apache Flink 的亚马逊托管服务的指标。	2016 年 12 月 1 日
增加了对百分位数统计数据的支持	您可以指定任何百分位数，最多使用两位小数 (例如 p95.45)。有关更多信息，请参阅 百分位数。	2016 年 11 月 17 日
已为 Amazon Simple Email Service 添加指标	已为 Amazon Simple Email Service 添加指标。	2016 年 11 月 2 日
更新的指标保留	Amazon CloudWatch 现在将指标数据保留 15 个月而非 14 天。	2016 年 11 月 1 日
更新的指标控制台界面	已使用对现有功能的改进和新功能更新 CloudWatch 控制台。	2016 年 11 月 1 日
已为 Amazon Elastic Transcoder 添加指标	已为 Amazon Elastic Transcoder 添加指标。	2016 年 9 月 20 日
已为 Amazon API Gateway 添加指标	已为 Amazon API Gateway 添加指标。	2016 年 9 月 9 日

更改	描述	发行日期
为 AWS Key Management Service 添加了指标	为 AWS Key Management Service 添加了指标。	2016 年 9 月 9 日
已为由 Elastic Load Balancing 支持的新 Application Load Balancer 添加指标	已为 Application Load Balancer 添加指标。	2016 年 8 月 11 日
添加了针对 Amazon EC2 的新的 NetworkPacketsIn 和 NetworkPacketsOut 指标	添加了针对 Amazon EC2 的新的 NetworkPacketsIn 和 NetworkPacketsOut 指标。	2016 年 3 月 23 日
已为 Amazon EC2 Spot 实例集添加新指标	已为 Amazon EC2 Spot 实例集添加新指标。	2016 年 3 月 21 日
已添加新的 CloudWatch Logs 指标	已添加新的 CloudWatch Logs 指标。	2016 年 3 月 10 日
已添加Amazon OpenSearch Service、AWS WAF 指标和维度	已添加 Amazon OpenSearch Service、AWS WAF 指标和维度。	2015 年 10 月 14 日
已添加对 CloudWatch 控制面板的支持	控制面板是 CloudWatch 控制台中的可自定义主页，可用于在单个视图中监控资源，即便是分布到不同区域的资源，也能对其进行监控。有关更多信息，请参阅 使用 Amazon CloudWatch 控制面板。	2015 年 10 月 8 日

更改	描述	发行日期
添加了 AWS Lambda 指标和维度	添加了 AWS Lambda 指标和维度。	2015 年 9 月 4 日
已添加 Amazon Elastic Container Service 指标和维度	已添加 Amazon Elastic Container Service 指标和维度。	2015 年 8 月 17 日
已添加 Amazon Simple Storage Service 指标和维度	已添加 Amazon Simple Storage Service 指标和维度。	2015 年 7 月 26 日
新功能：重启警报操作	增加了重启警报操作和与警报操作一起使用的新 IAM 角色。有关更多信息，请参阅 创建告警以停止、终止、重启或恢复 EC2 实例。	2015 年 7 月 23 日
已添加 Amazon WorkSpaces 指标和维度	已添加 Amazon WorkSpaces 指标和维度。	2015 年 4 月 30 日
已添加 Amazon Machine Learning 指标和维度	已添加 Amazon Machine Learning 指标和维度。	2015 年 4 月 9 日
新功能：Amazon EC2 实例恢复告警操作	更新了警报操作以包含 EC2 实例恢复操作。有关更多信息，请参阅 创建告警以停止、终止、重启或恢复 EC2 实例。	2015 年 3 月 12 日
已添加 Amazon CloudFront 和 Amazon CloudSearch 指标和维度	已添加 Amazon CloudFront 和 Amazon CloudSearch 指标和维度。	2015 年 3 月 6 日

更改	描述	发行日期
已添加 Amazon Simple Workflow Service 指标和维度	已添加 Amazon Simple Workflow Service 指标和维度。	2014 年 5 月 9 日
更新了指南以增加对 AWS CloudTrail 的支持	添加了一个新主题，说明如何在 Amazon CloudWatch 中使用 AWS CloudTrail 记录活动。有关更多信息，请参阅 使用 AWS CloudTrail 记录 Amazon CloudWatch API 和控制台操作。	2014 年 30 月 4 日
指南更新为使用新的 AWS Command Line Interface (AWS CLI)	AWS CLI 是一个跨服务 CLI，安装得到简化、具有统一配置且采用一致的命令行语法。Linux/Unix、Windows 和 Mac 均支持 AWS CLI。本指南中的 CLI 示例已更新为使用新的 AWS CLI。 有关如何安装和配置新 AWS CLI 的信息，请参阅 AWS Command Line Interface 用户指南中的使用 AWS CLI 界面进行设置。	2014 年 2 月 21 日
已添加 Amazon Redshift、AWS OpsWorks 指标和维度	已添加 Amazon Redshift、AWS OpsWorks 指标和维度。	2013 年 7 月 16 日
已添加 Amazon Route 53 指标和维度	已添加 Amazon Route 53 指标和维度。	2013 年 1 月 26 日
新功能：Amazon CloudWatch 告警操作	新增了一个区段用于记录 Amazon CloudWatch 告警操作，您可以使用此区段停止或终止 Amazon Elastic Compute Cloud 实例。有关更多信息，请参阅 创建告警以停止、终止、重启或恢复 EC2 实例。	2013 年 1 月 8 日
更新的 EBS 指标	已将 EBS 指标更新至包含配置的 IOPS 卷的两项新指标。	2012 年 11 月 20 日

更改	描述	发行日期
新账单警报	现在可以使用 Amazon CloudWatch 指标监控您的 AWS 收费并创建告警使其在超过指定阈值时通知您。有关更多信息，请参阅 创建账单告警以监控预估的 AWS 费用。	2012 年 5 月 10 日
新指标	现在可以访问六种新的 Elastic Load Balancing 指标，这些指标提供多种 HTTP 响应代码计数。	2011 年 10 月 19 日
新功能	现在可以从 Amazon EMR 访问指标。	2011 年 6 月 30 日
新功能	现在，您可以从 Amazon Simple Notification Service 和 Amazon Simple Queue Service 访问指标。	2011 年 7 月 14 日
新功能	添加了关于使用 `PutMetricData` API 发布自定义指标的信息。有关更多信息，请参阅 发布 自定义指标。	2011 年 5 月 10 日
更新的指标保留	Amazon CloudWatch 现在对告警历史记录的保留时间从之前的 6 周缩减到 2 周。这一改变让警报的保留时间和指标数据的保留时间相一致。	2011 年 4 月 7 日
新功能	添加了当指标越过阈值时发送 Amazon Simple Notification Service 或 Auto Scaling 通知的功能。有关更多信息，请参阅 告警。	2010 年 12 月 2 日
新功能	多种 CloudWatch 操作现在包括 MaxRecords 和 NextToken 参数，让您可以控制要显示的结果页。	2010 年 12 月 2 日
新功能	该服务现已集成 AWS Identity and Access Management（IAM）。	2010 年 12 月 2 日